Interactive Functions in Broadcast Video-on-Demand

by

MA Hon Shing

A dissertation submitted in partial fulfillment of the requirements
for the degree of Master of Philosophy
The Hong Kong Polytechnic University

Supervisor: Dr. To Tsun Ping Jimmy

Department of Electronic and Information Engineering
The Hong Kong Polytechnic University
Hong Kong SAR, China

This work was supported in part by
The Hong Kong Polytechnic University project G-V738

May 2002

Abstract of thesis entitled "**Interactive Functions in Broadcast Video-on-Demand**" submitted by Ma Hon Shing for the degree of Master of Philosophy at The Hong Kong Polytechnic University in May 2002.

Broadcast schemes for Video-on-Demand (VoD) service are done according to a predetermined schedule. New clients have to wait for the beginning of the next instance of the target video stream. A video server may not be able to deviate from its scheduled video data retrievals when a video stream is shared by more than one client. With this difficulty, the possibility of providing interactive VCR (video cassette recorder) functions in broadcast VoD has not been well addressed. In this project, two broadcasting schemes are proposed and analyzed to address the issue of VCR functions in broadcast VoD service. Each scheme has its own merits in supporting interactive functions. The first scheme, called Skip-Forward Broadcasting (SFB), allows clients to reposition their points-of-play in a forward direction. The second scheme, called Short-Range Fast-Forward Broadcasting (SRFFB), is designed to support clients to view the video at a doubled speed in the forward direction. Further, both schemes support pause/resume interaction and 'on the fly' seamless channel transition without any user-observable disruptions. However these two schemes have a large requirement on the client-side buffers. To reduce the buffer requirement, two additional schemes are also introduced. The third scheme, called Mirrored-Pyramid Broadcasting (MPB), supports fast-forward with a reduced client-side I/O bandwidth requirement. The fourth scheme, called Small-Buffer Skip-Forward Broadcasting (SB-SFB), is a modified version of the Skip-Forward Broadcasting scheme that reduces the buffer requirement by half.

Acknowledgement

I would like to express my deep gratitude to my supervisor, Dr. Jimmy To, for his persistent supervision, invaluable advice and indispensable help throughout my MPhil study. His kindness and patience have helped in one way or another to solve many challenging problems. I would also like to thank Dr. Daniel Pak-Kong Lun and Dr. Chi-Kwong Li for their advice and support.

In addition, I would like to thank my family for their long-term support. Thanks to my friends at the HKIBBS and the E-Fever BBS: Desmond, Helen, Chester, Cky and Edwin.

Finally, a very special thanks also to my soul mates, Dr. Pang Ka Lai, Mok Ka Yee and Tse Ka Yi. Thank you for taking time, time to stop and take an interest in me, time to listen to my problems and help me find the solutions, and most of all, time to smile at me and show you care. I do really enjoy our monthly gathering.

Ma Hon Shing

Table of Contents

List of Figures

List of Tables

List of Symbols

B	The network bandwidth allocated to a video in Mbps.
b	The normal video playout rate in Mbps.
K	The number of logical channels allocated to a video.
M	The playback duration of a video in minutes.
N	The total number of video segment blocks of a video.
X_j	The jth video segment block of a video.
$\|X\|$	The playback duration of a video segment block.
X^K_j	The jth video segment block of a video under a broadcasting scheme with K logical channels.
S_i	The video segment on logical channel i.
S^K_i	The video segment on logical channel i of a broadcasting scheme with K logical channels.
Q_i	The recurring period of S_i.
$C(i)$	The number of video segment blocks that made up the video segment on logical channel i.
α	The base number of logical channels allocated to a video.
U	A set of channel allocations with the same α setting.
U^K	The K-channel allocation scheme in a set U.
U^K_i	The logical channel i of the K-channel allocation scheme in a set U.
β	The ratio between the sizes of S_i and S_{i+1}.
B'	The bandwidth of a logical channel in Mbps.

Chapter 1: Introduction

1.1 Video-on-Demand

Video-on-Demand (VoD) is an electronic form of video rental service. It offers clients a video library from which they can select a video programme to watch at any time. Further, Interactive VoD allows clients to control the progress of a video using VCR (video cassette recorder) functions such as pause, resume and fast-forward.

VoD service represents a fundamental change in the TV watching paradigm. In the traditional broadcast TV system, many stations broadcast their programs simultaneously, and a client can choose to tune onto any specified TV channel. As a result, the client is allowed to select from a group of parallel and competing TV programmes [22]. Additionally, if a client misses the starting time of a programme, he/she will need to wait for another opportunity, which can be hours in the case of a cable service, or weeks in the case of wireless broadcast TV channels. In contrast, VoD service makes all programmes available to the customer without such restrictions.

Basically, VoD service can be classified into two quality-of-service categories, namely True VoD and Near VoD. True VoD needs to provide instant-starting service with full VCR functions. In Near VoD service, the instant-start and VCR requirements are normally relaxed to achieve cost-effectiveness.

1.2 VoD System Architecture

A typical VoD system architecture consists of a local video server connected to clients via a communication network. The client's hardware consists of a customer interface device (also called set-top box) with network interface, a display and a remote control unit for interacting with the system. Figure 1.1 illustrates this architecture.

This system can be part of a hierarchical video distribution system which consists of local and remote video servers [23]. The video servers are connected through a high-speed network. Information is cached locally and subsequently delivered to clients. Such a video

distribution scheme serves many purposes [22]. First, it increases availability and reliability of the system. Secondly, the provider can tailor the information delivery to a user group in a particular geographic area. Finally, it is easier to manage as each local system is responsible for its own billing and accounting. Now let us examine in greater detail the different components of this VoD architecture.

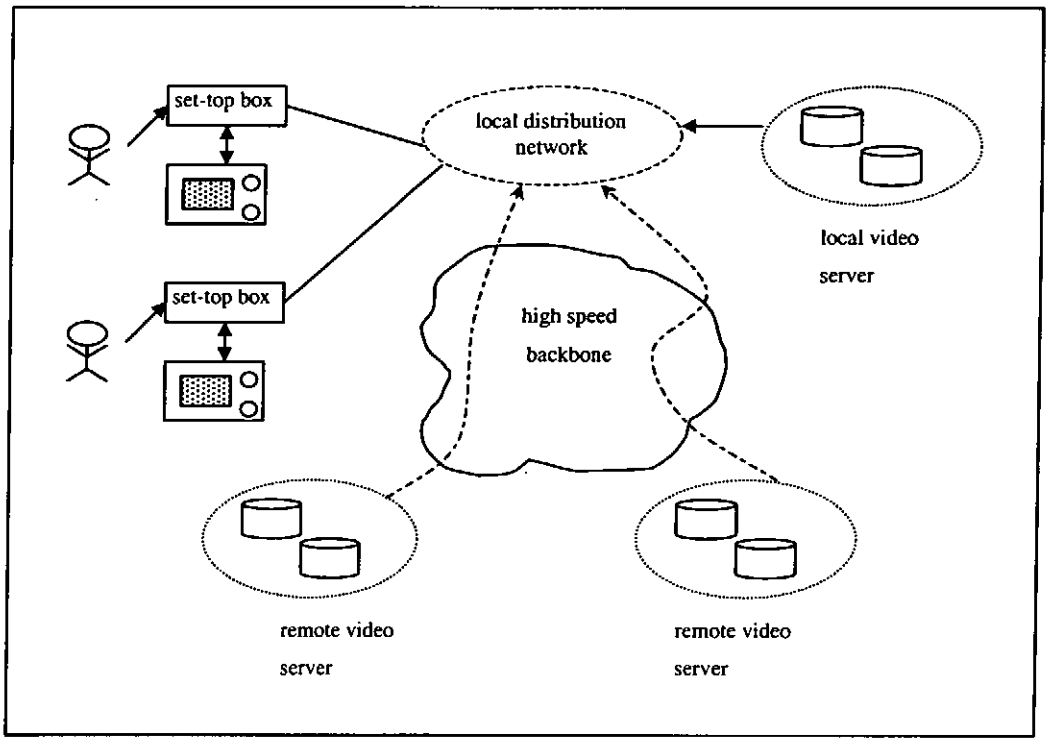

Figure 1.1. A simple VoD architecture.

1.2.1 VoD Server

VoD servers are responsible for processing clients' requests and maintaining the data streams used to deliver requested videos. To support a large client population and a wide variety of videos, a VoD server has to store and manage large volumes of data. It is likely that the videos are stored digitally in a compressed form to save storage space and to reduce transport cost. For example, a 120-minute video, using MPEG-2 compression (average rate 4 Mbps), requires about 3.6 GB of storage. Since video files have different popularities, they have to be well allocated in the servers in order to reduce the storage costs [7, 19, 28, 35].

A VoD server typically uses fast harddisk drives to deliver video streams to the clients. As the total disk bandwidth is a large multiple of the video playout rate, each disk can support a number of simultaneous video streams. The server maintains memory buffers and refills

them successively by reading ahead from the video files. Each retrieval of video data from the disk involves disk seek and rotational overhead times [29]. On the other hand, the real-time characteristic of video playout requires the VoD servers to observe strict deadlines. Any deviation from the timing sequence can lead to jerky motion. Hence, it is obvious that the time a disk takes to refill the data buffers for all the streams has to be less than the time it takes to exhaust the buffered readahead data. This constraint limits the maximum number of simultaneous streams that a server can accommodate.

In order to maximize the number of concurrent video streams, disk scheduling is employed in VoD servers to minimize the service latency between video streams, as well as the latency between successive services for a video stream [10, 30, 36]. Since the video streams are served in rounds, buffers are needed in the client-side to avoid output device starvation. The disk scheduling policy, therefore, will affect the client-side buffer requirement. Moreover, if VCR interactions are to be supported by a server, the number of concurrent video streams that can be supported will be less. For instance, in the fast-forward playout, video data are consumed at a faster rate. The disk, therefore, needs to read more for that video stream so as to prevent buffer underrun. As a result, the total number of video streams that can be supported in each round is decreased.

The server has a finite amount of resources. Therefore, it has to employ admission control algorithms to decide if a particular customer request can be satisfied given the current system load and resources [21, 23, 33]. A simple admission control policy that allocates resources conservatively using the worst-case assumption may give a poor resource utilization. Alternatively, a statistical admission control, based on observation, can be employed. Statistical admission control will try to increase the number of clients until real-time constraints are marginally violated, at the expense of an appreciable degradation in the quality of service [33].

1.2.2 Communication Network

Like disk bandwidths in VoD servers, the bandwidth of the communication network is also finite. Therefore, it has to be utilized properly. Even though clients may access the VoD system randomly, having a priori knowledge about user access patterns can lead to efficient network bandwidth management. For example, if the traffic characteristics indicate that a

video is popular at a particular site, the system can cache the video locally to increase availability.

The pattern of user access to the system is unlikely to be uniform over any given 24-hour period. Typically, the load would be low to moderate during daytime, increase gradually through the evening, and fall off during the night. Figure 1.2 illustrates a hypothetical access pattern to the VoD system. The loading in the system is high during evening hours, peaking at around 8:00 PM, and is low to moderate at other times. Similar models can be built for different geographical regions, video types and individual titles. Such models can be used to update the video popularity tables, redistribute data, as well as reconfigure the system during off-peak hours.

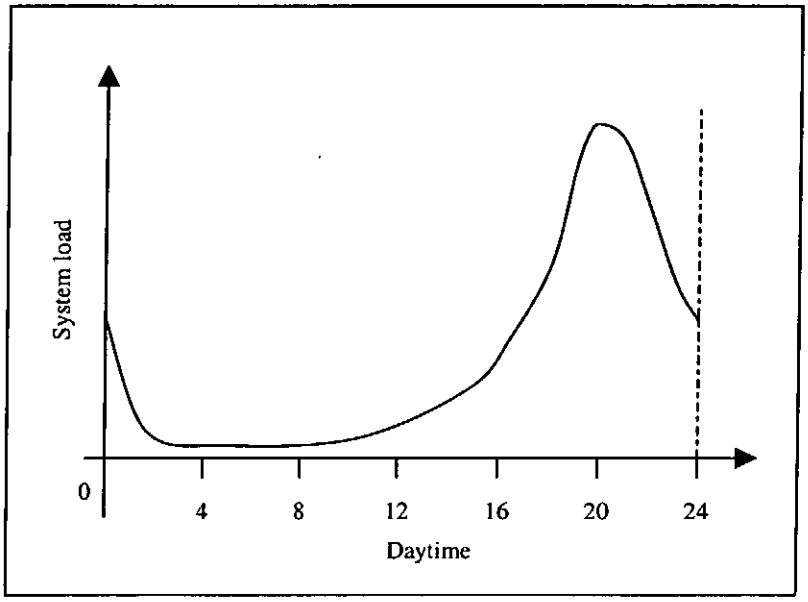

Figure 1.2. A hypothetical daily-access model for a VoD system.

1.2.3 End-user Subsystem

In the end-user subsystem, there exists a device through which the client can interact with the VoD servers. This device is called customer interface device, also known as set-top box (STB). The STB decodes incoming data and delivers a video image to the display. It also translates the client's interactions to give appropriate messages to the server.

Normally, a buffer inside the STB is present for smoothing of variable network delivery latencies. For traditional broadcasting and VoD systems with little or no buffering, a client and its server are tightly coupled. Any change in the consumption of video data from the

client has to be immediately and continuously handled by the server. With buffering, changes in consumption do not require instantaneous adjustment. Larger buffers allow greater latitude in handling these adjustments, especially during VCR interactions.

An important consideration in the evaluation of a VoD system is the cost of the STB since the device needs to be provided for each client. The cost of the STB needs to be minimized to make it technologically cost-effective and commercially competitive.

1.3 Performance Objectives

In VoD systems, the performance objectives to be met are:

- Use the smallest server-side disk I/O and network bandwidths to support the largest client population.
- The ***startup latency*** (the time latency between the client's request arrived at the system and the data is delivered to the client's display device) should be minimized.
- VCR interactions are provided by the system.
- Use the smallest storage to provide a large variety of videos.

It is obvious that some of the objectives listed above are conflicting. For instance, to provide VCR interactions, the server-side disk I/O and network bandwidths need to be increased in order to serve the same client population. To provide instant starting, the server needs to support a large number of concurrent streams for each video such that the client can capture one without excessive waiting time.

1.4 Service Policies

The number of clients that the server can simultaneously support depends on how the server allocates video streams to arriving requests from the clients. The scheduling and the allocation of video steams give rise to different service policies [34]. The VoD server can choose to start a video stream as it is requested by a client, or at fixed regular intervals. Further, each video stream can be dedicated to a single client or shared by a pool of clients. The combination of these criteria results in three service policies which are discussed below.

The interested reader is referred to [11] for a comprehensive overview of several major service policies.

1.4.1 Unicast

In unicast VoD systems, clients are individually allocated and dedicated a transmission channel and a set of server resources. This service policy can be considered as True VoD service. The client is served immediately upon request. However, clients need to wait whenever the server reaches full capacity. Since only one client is the user of each video stream, the implementation of VCR interactions is easy because the server does not need to cater for the impact of these interactions on the other clients.

However, this service policy leads to an expensive-to-operate and non-scalable system. This is because the number of simultaneous clients that can be supported is limited by the servers' capacity. The startup latency can easily be very long once the server runs out of free resources. Employing a larger number of storage devices can proportionally yield more video streams only if the mapping of clients' video choices to available storage is ideal. In other words, a selected video may not be served by any arbitrary storage device in the system. In reality, a client's selection can only be retrieved from the disks where a copy of the selected video is stored. If an imperfect mapping arises, the imbalance in load distribution over the server's disk drives will also prevent optimum utilization [22].

1.4.2 Broadcast

In this service policy, each video stream is shared by an unbounded number of clients [2, 13, 15-18, 24-25, 32]. For each video, a new data stream is started at regular intervals. All the clients who request for the same video will be grouped or batched together and simultaneously served by the same new data stream. Thus, clients will have to wait for the broadcast of the next instance of the video stream before they can view the video. The startup latency depends on the video's length and the number of video streams that the VoD server has allocated to the video. For example, if a 120-minute video is allocated ten streams, the longest startup latency will be twelve minutes.

Although this service policy increases the number of clients that can be served simultaneously with little server disk I/O bandwidth, the implementation of VCR interactions is more complicated than that in the unicast policy. This is because the sharing of video

streams does not allow the server to deviate from its scheduled retrieval of video data. The video server cannot alter the data retrieval of a video stream to retrieve a video block in response to a random access request from one client, as the video stream may be shared by other clients at the same time. With this difficulty, the provision of interactive VCR functions in broadcast VoD systems has not been well addressed.

Broadcasting is good for cable and microwave transmission technologies in which bandwidth is the most precious resource. The most related works on broadcast VoD will be reviewed in more details in Chapter 2.

1.4.3 Multicast

This service policy addresses some of the drawbacks of the previous two polices. In multicasting, a video stream is also shared by a pool of clients. How a VoD server starts a video stream depends very much on the system design [5, 6, 27]. The server may start a new video stream when the number of waiting clients or the elapsed time reaches a certain threshold. Thus, the time skews between streams of the same video are not necessarily regular. To further increase the utilization of video streams, three approaches, namely Caching [4], Adaptive Piggybacking [12] and Patching [14], are proposed.

The basic idea behind Caching is to store the data already read by the previous video stream in memory set aside as cache and feed the next stream for the same video from the cache rather than from the disk storage. This avoids the need to initiate a new video stream from the disk. To be effective, Caching requires abundant buffer at the server. The viability of the method depends on the cost trade-off between memory buffer and disk bandwidth. Moreover, these techniques often assume that spare network bandwidth is available for delivery. This is true in multicasting but is not necessarily valid in broadcasting.

The main idea behind Adaptive Piggybacking is to alter the playout rates (slow down or speed up) of the video streams (for the same video) in progress so that the differences in the points-of-play of nearby video streams diminish after some time. The clients served by these video streams can then be served by one single video stream. The advantages of this approach are that no buffer is needed during the merging and the new clients need not wait for a pre-fixed time for their videos to start, i.e., startup latency is shortened. However, the variation of the playout rate can only done to an extent that it is not perceptible by the client.

As a result, the number of streams that can be merged may be limited in the case of less popular videos.

Instead of altering the video playout rate, Patching increases the buffer space at the client-side for prefetching in order to improve the efficiency of multicasting. In Patching, a new client prefetches the future video data from an existing video stream (called *regular-channel*) while being served by a new video stream (called *patching-channel*). Once the point-of-play comes to a skew point [14], service can be switched from the *patching-channel* to the *regular-channel* which is shared by more clients (the skew is absorbed by the client-side buffer). Thus, the *patching-channels* are temporary and are used during the initial catch-up period. The effectiveness of Patching depends on the number of *regular-channels* and the client-side buffer size.

In the support of VCR interactions, a multicast service policy often suffers from the same problems as the broadcast policy in that one client's VCR interaction affects all other clients sharing the same data stream. The support of VCR interactions in the multicast scheme has been studied to some extent [1, 3, 20, 26]. In these studies, each VCR interaction needs to be supported by a new video stream. Hence, some streaming capacities of the server need to be reserved for the VCR interaction. In addition, network bandwidth, which is also needed to be reserved for the delivery of extra streams, is too expensive in broadcasting. Thus, these solutions in providing VCR functions cannot be applied to broadcasting.

1.5 Research Focus

As demands on different videos can hardly be uniform, some videos are more popular than others. It has been shown that the access frequencies to various videos can be characterized by a Zipf distribution [6]. Practically, most of the demand (80%) is for a few very popular videos. In the previous section, we have discussed the merits of different VoD service policies. It can be seen that clients requesting the same popular video thus are most economically served by repetitive broadcasting of overlapped instances at regular intervals. This cost-advantage has motivated us to focus on the VoD broadcasting services on popular videos.

In the design of a broadcast scheme, the following four issues need to be resolved:

1) Channel design

The collection of resources required to deliver a data stream is referred to as a ***logical channel***, or simply a ***channel*** [5]. The key resources include the disk I/O bandwidth in the server and the transmission bandwidth in the network.

The channel design deals with determining the transmission rate of a logical channel and how many logical channels are allocated to a video.

2) Segmentation design

Segmentation is the process of dividing a video file into small ***video segment blocks***, or ***blocks*** in brief. The size of the blocks can be uniform or different.

3) Broadcast schedule design

This determines how many blocks are put on each logical channel and how frequently these blocks are broadcast on the channel. Each series of blocks on a logical channel is called a ***video segment***. In other words, each channel has only one video segment consisting of one or more blocks. How these blocks made up the video segments on the channels is referred to as an ***allocation series***.

4) Playback strategy design

This refers to how a client starting and acquiring the desired video data from the logical channels in order to enjoy continuous viewing. The way a client collects/downloads the video data from channels is called a ***download sequence*** and the timeslot in which the first segment of a video is downloaded is referred to as the ***starting timeslot***. Different starting timeslots may result in different download sequences.

Performance objectives more specific to broadcast schemes are as follows:

- The network bandwidth allocated to a video can be adjusted by the service provider dynamically in order to adapt to demand changes.
- In the client's set-top box, the buffer requirement is to be minimized in order to reduce cost.

- VCR interactions are provided.
- The startup latency is to be small.
- The client-side I/O bandwidth requirement is the minimum.

Unfortunately, the VoD system has finite resources measured in terms of storage I/O and network bandwidths. Since customers compete for the same resources, system developers have to design efficient schemes that ensure fair allocation. At the same time, service providers want to generate maximum profits from the offered services. We need to balance these two conflicting requirements in order to minimize the cost and maximize the potential revenues of the system. The objective of this research is, thus, to explore approaches that can achieve significant improvements in one or more performance objectives.

In the next chapter, previous works on broadcast schemes will be examined and discussed. Original work will be discussed in Chapter 3 to 6. The first scheme, called Skip-Forward Broadcasting scheme which allows clients to reposition their points-of-play, will be introduced and analyzed in Chapter 3. In Chapter 4, another scheme called Short-Range Fast-Forward Broadcasting scheme will be discussed in detail. The last two schemes, Mirrored-Pyramid Broadcasting scheme and Small-Buffer Skip-Forward Broadcasting scheme will be presented in Chapter 5 and Chapter 6, respectively. The latter two aim to reduce the client-side buffer requirement.

Chapter 2: Related Works

In this chapter, we shall review some broadcasting schemes proposed by researchers within our specific area of focus. All of them aim to economically broadcast popular videos to clients. Without loss of generality, it is assumed in this chapter that the system has only one video. The list of symbols used in this report can be found on page *vii*.

2.1 The Phase-Constrained Allocation (PCA) Scheme

In [24], Özden et al. proposed a low-cost storage hierarchy which supports a large number of client requests. A video is modeled as a two-dimensional array of video segment blocks as shown in Figure 2.1*a*. Each phase (or row) represents a different point-of-play of the video and is further viewed as consisting of n video segment blocks each of size d, where d is the media unit retrieved from disks each time.

The video segment blocks are laid out on the disk in a column major form (Figure 2.1*b*), such that retrieving video data sequentially from the disk enables blocks of different phases to be retrieved concurrently. On the other hand, clients view the video in a row-major order (Figure 2.1*c*). The transmission rate of video data from the server to clients is equal to the normal playback rate of the video. As a result, the size of client's buffer is as small as one block. The time taken for the disk head to read from the first column to the last column such that a new video stream can be initiated is the startup latency.

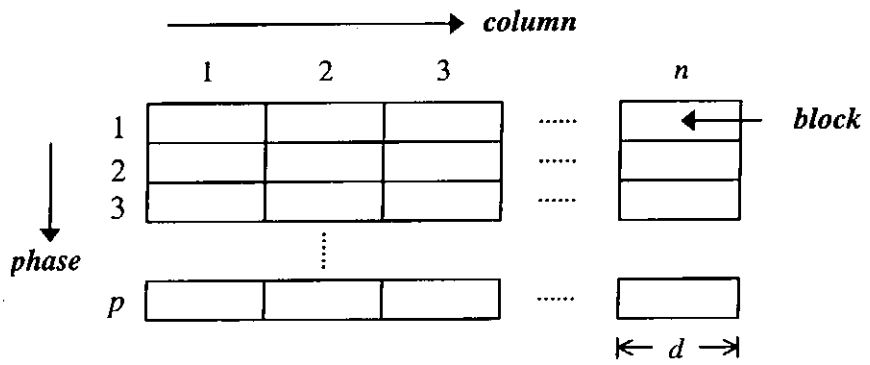

Figure 2.1a. Logical view of a video in PCA.

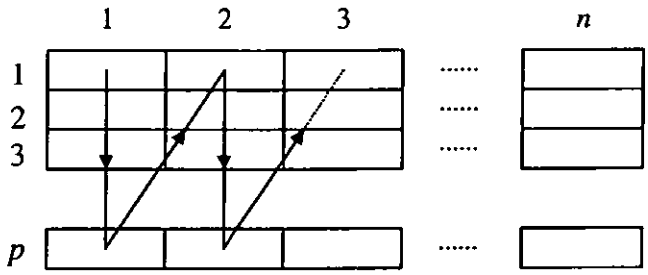

Figure 2.1b. Retrieval sequence (column-major order) in PCA.

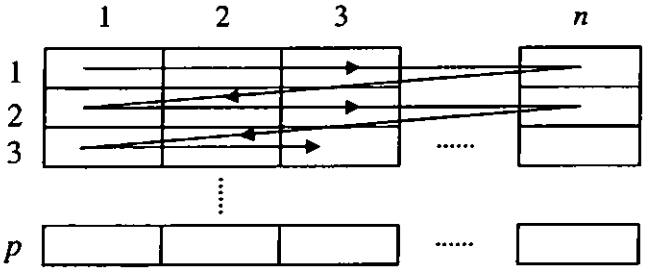

Figure 2.1c. Viewing sequence (row-major order) in PCA.

If this approach is used to broadcast a video over p channels, the startup latency can only be improved linearly with the increase in the network bandwidth. Moreover, the support of VCR functions in the scheme needs extra network bandwidth and thus is not suitable for cable or microwave transmission technologies.

2.2 The Pyramid Broadcasting (PB) Scheme

Back in 1996, a new broadcasting scheme called Pyramid Broadcasting [32] for supporting frequently requested videos was proposed. The objective of PB is to reduce the startup latency without the need to increase the allocated network bandwidth proportionally.

The basic idea of PB is to break up the video file into video segments of increasing size. Each segment, composed of one or more uniform video segment blocks, is broadcast on one logical channel. To ensure that a video can be viewed continuously, the segments are broadcast in a way such that the time to start downloading of video segment $(i + 1)$ is not later than the exhaustion time of video segment i. The startup latency of a video decreases with the size of the first segment. By minimizing the size of the first segment, PB can broadcast the first segment more frequently on the channel and consequently reduce the startup latency.

In PB, the network bandwidth B allocated to a video is divided into K logical channels each having bandwidth B' (where B' is equal to B/K). B and B' are in multiples of the video playout rate. Each M-minute video is also divided into K segments. The size of segment $(i + 1)$ is made β times that of segment i. Finally, segment i is put on channel i and is broadcast periodically.

On the client side, downloading begins with the first segment at the first occurrence, while consumption also starts concurrently. For subsequent segments, the client downloads the following segment at the earliest possible time after playback of the current segment has begun. At any one time, the client downloads from at most two consecutive channels. Thus, the parameter β needs to be chosen in such a way to ensure that the playback duration of the current video segment is longer than the worst latency in downloading the next segment.

The startup latency of a video is equal to the time between successive broadcast instances of the first segment. The startup latency is in PB is $\frac{M*(\beta-1)}{\beta*(\beta^K-1)}$ and is roughly reduced by a factor of β for each addition channel. The optimal startup latency can be obtained by using a value of β around Euler's constant (e = 2.72) [32]. In terms of client-side buffer requirement, PB requires each client to have a buffer of size $\left(\frac{M*(\beta^{K-2})*(\beta-1)}{(\beta^K-1)}\right)*b$, where b is the playout rate of the video.

Figure 2.2 shows the startup latency and the client-side buffer requirement of PB under different number of channels allocated to a 120-minute video. It can be seen that PB can improve the startup latency exponentially with respect to the allocated network bandwidth. However, it requires quite a large buffer size (>65% of the video file size) at the client's set-top box.

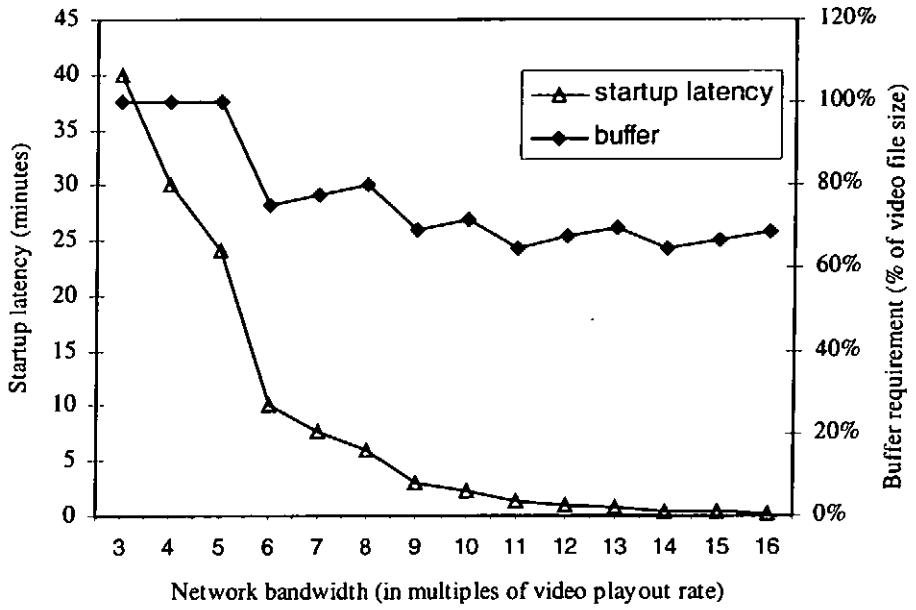

Figure 2.2. The startup latency and the client-side buffer requirement of PB.

2.3 The Fast Data Broadcasting (FDB) Scheme

Although the Pyramid Broadcasting scheme improves the startup latency exponentially with respect to the allocated network bandwidth, it requires quite a large buffer size. This buffer requirement does not significantly change with the allocated bandwidth. In [18], Juhn proposed a broadcasting scheme called Fast Data Broadcasting (FDB) that can reduce the client-side buffer requirement to 50% of a video file size.

In FDB, a video is allocated K logical channels where the bandwidth of each channel is equal to the video playout rate. A video is then equally divided into N video segment blocks, where

$$N = \sum_{i=0}^{K-1} 2^i = 2^K - 1$$

Next, 2^i continuous blocks are allocated into channel i and are broadcast periodically as shown in Figure 2.3. In the figure, the numbers in the slots are the block numbers. Hence, the number of blocks on channel $(i + 1)$ is twice of that on channel i.

channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3
channel 2	4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7

Figure 2.3. The broadcast schedule of FDB (3 channels).

Similar to PB, the client begins downloading the first segment at the first occurrence and starts to consume it concurrently. However, the client may have to download blocks from all the remaining logical channels concurrently to ensure a continuous viewing of the video as illustrated in Figure 2.4 where the shaded blocks need to be downloaded.

channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	3	2	3	2	3	2	3	2	3	2	3	2	3	2	3
channel 2	4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7

Figure 2.4. The worst-case concurrent download in FDB.

In FDB, the startup latency for an M-minute video is equal to the playback duration of the first segment, which is $\frac{M}{2^K - 1}$. The client-side buffer requirement is $(1 - \frac{2^{K-1}}{2^K - 1})Mb$ (b is the normal playout rate of the video), or roughly 50% of a video file size. This is a saving up of 23% relative to PB. Figure 2.5 shows the startup latency of FDB and PB. It can be seen that FDB has a better performance in the startup latency than PB.

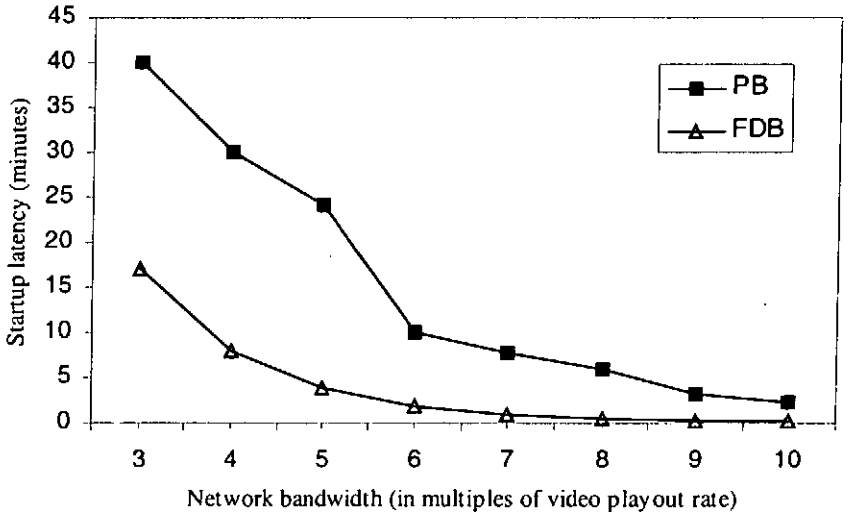

Figure 2.5. The startup latencies of FDB and PB.

2.4 The Seamless Channel Transition (SCT) Scheme

The demand of a video may be affected by factors such as time of a day, day of a week, or special holidays. Even for popular videos, the demand within a day is not uniform. Thus, it is desirable to dynamically adjust the number of logical channels allocated to a video on-the-fly such that the network bandwidth can be utilized efficiently. However, the broadcasting schemes we have reviewed all aim at reducing the startup latency given a *fixed* bandwidth for a video and cannot dynamically adapt to demand changes.

In the Seamless Channel Transition (SCT) scheme proposed by Tseng [31], the number of channels allocated to a video can be dynamically adjusted seamlessly such that during the transition period, clients currently viewing the video will not experience any disruption. Meanwhile, new clients can be accepted and served immediately. Tseng's design is based on the Fast Data Broadcasting scheme. Thus, it inherits the basic features of FDB. The main difference is that, dummy video data blocks need to be padded at the end of the video in SCT.

For FDB with K logical channels, the number of blocks is $(2^K - 1)$. Thus for an M-minute video, the size of a block is in the number series [M/1, M/3, M/7, M/15, M/31, ...] as the number of channels increases. Since the denominators in the series are mutually prime, there is no clear relationship between the size of blocks from any two FDB with different number of channels. To solve this problem, SCT uses a data padding method.

To carry out data padding, a base number of channels allocated to a video, denoted as α, needs to be decided first. This is the minimum number of channels that needs to be allocated to the video in the scheme. Once α is chosen, the size of dummy video data for an M-minute video can be determined by the following equation:

$$Video_{dummy} = \frac{M}{2^{\alpha} - 1}$$

Figure 2.6 shows some segmentations in SCT with $\alpha = 2$ under different number of channels. In SCT, the padded video is divided into 2^K blocks instead of $(2^K - 1)$ blocks.

Number of channels	The new video file															
	Original video file												Dummy video data			
2	1				2				3				4			
3	1		2		3		4		5		6		7		8	
4	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 2.6. The segmentations of a video in SCT ($\alpha = 2$).

After data padding, the new video file length becomes

$$M' = M + \frac{M}{2^{\alpha} - 1}$$

$$= \frac{2^{\alpha} * M}{2^{\alpha} - 1}$$

Since SCT is built on top of FDB, the number of blocks to be broadcast also follows the number series [1, 3, 7, 15...] as the number of channels increases. As a result, the last block, which is purely dummy video data, is never broadcast on any channel.

To perform channel transition, say between SCT with K and $(K + 1)$ channels and $K \geq \alpha$, the ending times of the respective first video segments need to be aligned first. By doing so, Tseng proved that the video data of the K-channel scheme at timeslot t can be found in the corresponding timeslots in the $(K + 1)$-channel scheme.

There are two types of channel transition, ***step-up*** and ***step-down***. In step-up transition, SCT can finish the operation instantaneously and seamlessly. In step-down transition, the number of logical channels is decreased after the transition. This means that the video data that can be broadcast in one timeslot is reduced such that some video data scheduled to be delivered in the old broadcast schedule will not be broadcast in the new stepped-down broadcast schedule. These 'missing' video data may cause viewing disruption to the clients. To solve this problem, Tseng suggests that the missing video data can be packed together and then broadcast on the channels that are going to be returned back to the VoD system. Such to-be-returned channels can only be released after the delivery of the missing video data is completed. Therefore, step-down transition in SCT cannot be immediate and takes considerable time to complete.

2.5 The Skyscraper Broadcasting (SB) Scheme

Similar to the previous reviewed broadcasting schemes, the Skyscraper Broadcasting (SB) scheme also fragments a video into segments of increasing sizes and broadcast them on separate channels periodically at the video playout rate. The main difference between SB and other schemes is the growth pattern of segment size.

Instead of monotonic increasing the size of every segment, except for the first segment, every subsequent pair of consecutive segments of SB has the same size. To prevent the segments becoming too large, SB also sets a restriction factor, denoted as W, such that the size of segments will not exceed W blocks. The allocation series of SB follows the function below:

$$C(i) = \begin{cases} 1 & i = 1, \\ min(W,\ 2) & i = 2, 3, \\ min(W,\ 2*C(i-1) + 1) & i \bmod 4 = 0, \\ min(W,\ C(i-1)) & i \bmod 4 = 1, \\ min(W,\ 2*C(i-1) + 2) & i \bmod 4 = 2, \\ min(W,\ C(i-1)) & i \bmod 4 = 3 \end{cases}$$

where $C(i)$ is the number of blocks that made up the segment on channel i. The growth pattern is (1, 2, 2, 5, 5, 12, 12, ...). From the allocation series of SB, we can see that every two channels form a phased-constrained allocation while the overall SB is a pyramid-based broadcasting scheme. Therefore, it can be said that SB is a hybrid of PCA and PB. Because of this, SB has inherited characteristics from both PCA and PB.

For SB with K channels, the startup latency for an M-minute video is $\frac{M}{\sum_{i=1}^{K} min(C(i), W)}$ and the buffer requirement is $(W - 1)$ blocks. Figure 2.7 and Figure 2.8 show, respectively, the buffer requirement and startup latency of SB with different values of W. With unrestricted W, the buffer requirement, in fraction of a video file size, is the smallest (20%) when the number of channels is three. For other number of channels, the buffer requirement ranges from 26% to 40%. From Figure 2.7, we can observe that the buffer requirement of SB with restricted W is much smaller than that of unrestricted W. However, their performance in the startup latency does not scale as well with respect to the allocated bandwidth. With restricted

W, more channels are needed in SB so as to provide small startup latency because the rate of growth is constrained. Therefore, SB achieves a small buffer requirement at some trade-off between the buffer requirement and the startup latency/the network bandwidth. Figure 2.8 also compares the performance in startup latency between FDB and SB. FDB has a much smaller startup latency than SB because it has a quicker growth in the number of blocks and can minimizes the size of the first segment.

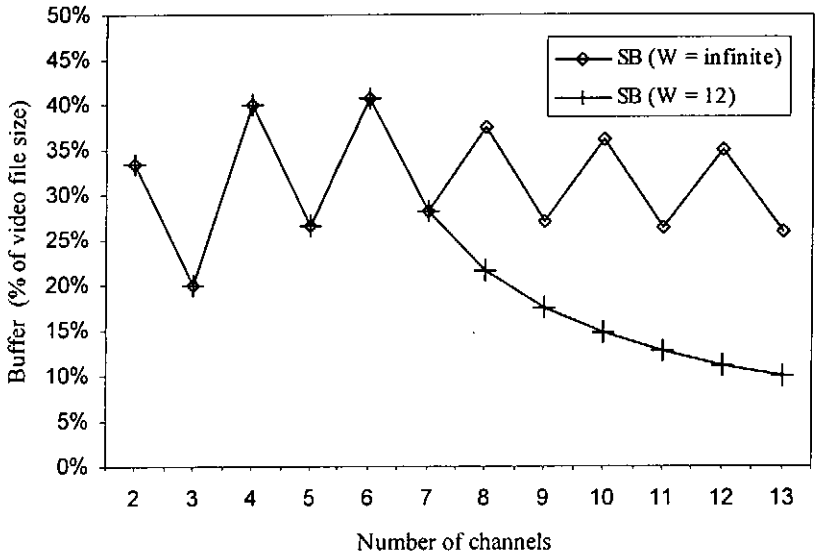

Figure 2.7. Buffer requirements of SB (*W* = infinite, 12).

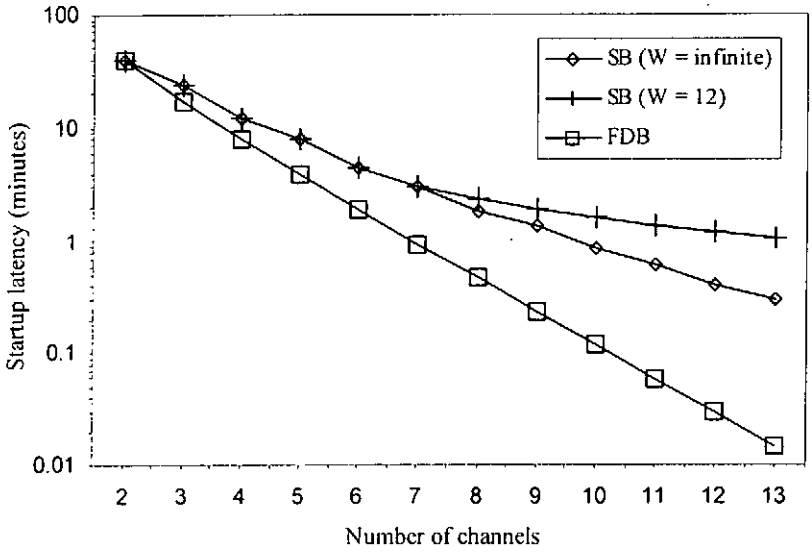

Figure 2.8. Startup latencies of SB (*W* = infinite, 12) and FDB.

A dynamic Skyscraper Broadcasting scheme was proposed in [8] to improve the performance of SB for lower client request rates and time-varying video popularities. In [9], Eager et al. further shows that the dynamic SB can be applied in multicast VoD systems.

2.6 Zero-Delay Broadcasting Protocols

In [25], Pâris proposed a technique called *Partial Preloading* which can archive a zero startup latency. Pâris observed that many broadcasting schemes require a large client-side buffer size and use one dedicated channel to deliver the first few minutes of a video in order to reduce the startup latency. Therefore, Pâris suggests preloading the first few minutes of the top ten to twenty most popular videos (that are likely to be requested by over 40% of all clients) into the client's buffer so that clients can start these videos instantaneously. Preloading can also reduce the network bandwidth required to broadcast these leading video data. Pâris also presented two new schemes with partial preloading in [25], namely Polyharmonic Broadcasting with Partial Preloading and Mayan Temple Broadcasting. However, both schemes assume that the client-side buffer has enough space to store at least 40% to 50% of the video file.

2.7 VCR Interactions

In the previous sections, several broadcasting schemes are reviewed. Both the Pyramid Broadcasting scheme and the Fast Data Broadcasting scheme can improve the startup latency exponentially, while FDB has a smaller client-side buffer requirement. The Seamless Channel Transition scheme can dynamically adjust the number of logical channels according to demand changes. Besides, the Skyscraper Broadcasting scheme can archive a very small buffer requirement at the expense of more channels. Last but not least, Partial Preloading can help to archive zero startup latency for popular videos. However, none of these schemes has addressed the issue of VCR interaction. In the following chapters, several new broadcasting schemes which have different merits in supporting VCR interactions will be proposed and analysed. FDB will be mainly used as a yardstick for comparison because it has a balanced performance in startup latency and client-side buffer requirement. In addition, FDB can support seamless channel transition after being modified to SCT.

Chapter 3: Skip-Forward Broadcasting Scheme

3.1 Skip-Forward Interaction

In this section, we shall discuss how skip-forward interaction can be supported in broadcast VoD service. In ***skip-forward*** interaction, a viewer is allowed to reposition his/her point-of-play in a forward direction instantly. In the known broadcasting schemes, a video is quantized into video segment blocks which are delivered on logical channels. Thus, skip-forward interaction in broadcast schemes can be simulated by repositioning the point-of-play to a point in the succeeding video segment block, provided that the downloading of the succeeding block always is in progress or completed.

There are two modes of download in broadcasting schemes. The first mode is ***concurrent downloading*** from several logical channels. In this mode, some blocks are prefetched in the buffer. Another mode is ***just-in-time downloading*** where the same block is being viewed while it is being downloaded, which means that no blocks are readahead. These two modes can be mixed in a download sequence. However, the broadcast schedules inherent in a scheme chosen will impose constraints on which of the above download modes can be used.

In the Fast Data Broadcasting (FDB) scheme [18], the download sequence of video segment blocks is usually performed in an optimistic way. In a download sequence, there can be more than one feasible timeslot to download a particular block. The later timeslot with the largest timestamp is to be chosen to reduce the client-side buffer requirement. Such an optimistic approach allows just-in-time downloading. Under this approach, a just-in-time downloading subsequence may occur. If the subsequence begins with the first segment of the video, it is referred to as a ***Just-In-Time Sequence (JITS)***. The length of such a sequence depends on the starting time, which may vary. The main characteristic of a JITS is that no block in the sequence is readahead. Therefore, if a subsequence of just-in-time downloading occurs in the middle of the download sequence but not at the beginning, it is not classified as a JITS because some blocks are prefetched already before consumption. The JITS needs not occur throughout the download sequence. It is possible that within a complete download sequence, only the first part of downloading is a JITS. Figure 3.1 and 3.2 show two examples of JITS.

To perform skip-forward, the next video segment block with reference to the current point-of-play needs to be already prefetched or is currently broadcasting on a channel, otherwise the interaction cannot be performed. JITS, thus, does not allow skip-forward interaction.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	**1**	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	3	**2**	**3**	2	3	2	3	2	3	2	3	2	3	2	3
channel 2	4	5	6	7	**4**	**5**	**6**	**7**	4	5	6	7	4	5	6	7

Figure 3.1. An example of Just-In-Time Sequence (shaded blocks) of FDB (3 logical channels).

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	**1**	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	3	2	3	**2**	**3**	2	3	2	3	2	3	2	3	2	3
channel 2	4	5	6	7	**4**	**5**	**6**	**7**	4	5	6	7	4	5	6	7

Figure 3.2. An example of partial Just-In-Time Sequence (shaded blocks) of FDB (3 logical channels).

JITS is caused by the implementation of optimistic downloading. It appears that if we change the download sequences into a greedy way such that all video segment blocks are downloaded immediately at their first occurrence (ignoring other feasible timeslots in the future), then the problem may be solved. However, this is not necessarily true. Even when the blocks in FDB are downloaded in the greedy way, just-in-time downloading may still occur.

Proposition 3.1: *For the first block of the video segment on any logical channel in FDB, except the first channel, there is one and only one suitable timeslot to download no matter which timeslot the download sequence starts with.*

Proof:

For FDB with K channels, a video is segmented into $(2^K - 1)$ video segment blocks. On channel i, a video segment which composed of 2^i video segment blocks is broadcast repeatedly. Denoting the video segment on channel i as S_i and the jth video segment blocks as X_j, we have

$$S_i = X_{2^i} \mid X_{2^i+1} \mid \ldots \mid X_{2^{(i+1)}-1}$$

The video segment S_i, or each block of the segment, recurs every 2^i timeslots on the channel. The duration of 2^i timeslots is referred to as the ***recurring period*** of S_i (or the blocks of S_i), and denoted as Q_i. For any two blocks, their difference in the playtime at the normal playout rate is defined as ***playtime difference***.

The first block of S_i is X_{2^i}. The playtime difference, in units of timeslots, between X_{2^i} and the first block of the video, X_1, is $(2^i - 1)$. Since $(2^i - 1) < 2^i$,

$$2^i - 1 < Q_i$$

As the playtime difference is smaller than the recurring period of X_{2^i}, it's possible that no instance of X_{2^i} starts within the playtime difference period. If no instances of X_{2^i} begins within the period, one instance must then exist in the timeslot just after the playtime difference period, i.e., just-in-time downloading needs to be performed. •

Proposition 3.2: *In a Just-In-Time Sequence of FDB, the video segment blocks, excluding the first block of the video segment on each logical channel, could be prefetched under the greedy download sequences of FDB.*

Proof:

The non first-video-segment-blocks of S_i are X_{2^i+y}, where $0 < y < 2^i$. The playtime difference, in units of timeslots, between X_{2^i+y} and X_1 is $(2^i + y - 1)$. Since $(2^i + y - 1) \geq 2^i$,

$$2^i + y - 1 \geq Q_i$$

This playtime difference is the playtime of X_{2^i+y} with reference to the consumption of X_1 at the normal playout rate. Since the difference is larger than the recurring period of the blocks on channel i, an instance of each non first-video-segment-block will be broadcast at least once within the playtime difference period and thus be prefetched. •

From the propositions, we can see that within a JITS in FDB, the just-in-time downloading problem of the first video segment block of each video segment on channels cannot be

resolved even by using a greedy download sequence. In order to provide skip-forward interaction, extra channels are needed to deliver these first blocks redundantly one timeslot before their broadcast schedule. To find out how many extra logical channels are needed, we need to know the probability of JITS arising in FDB. The probability is an inverse measure of the number of starting timeslots that one can find an instance of a just-in-time downloading problem.

Proposition 3.3: *In FDB, the probability of forming a Just-In-Time Sequence between an instance of a video segment on channel 0 and any instance of a segment on channel (i + 1) is* $(1/2)^i$.

Proof:

In FDB, the video segment on channel i consists of 2^i blocks. Now let us consider the segments on two consecutive channels, i and $(i + 1)$. The size of segment on channel $(i + 1)$ is twice of that on channel i. This means that two instances of S_i will be broadcast while one S_{i+1} is delivered. For every two deliveries of S_i, one will be aligned with a S_{i+1} while another will be aligned with the center of the same S_{i+1}, or said to be joined to the beginning of the next instance of S_{i+1}, as illustrated in Figure 3.3. In other words, there is a chance of 1/2 for a S_i to form a JITS with S_{i+1}.

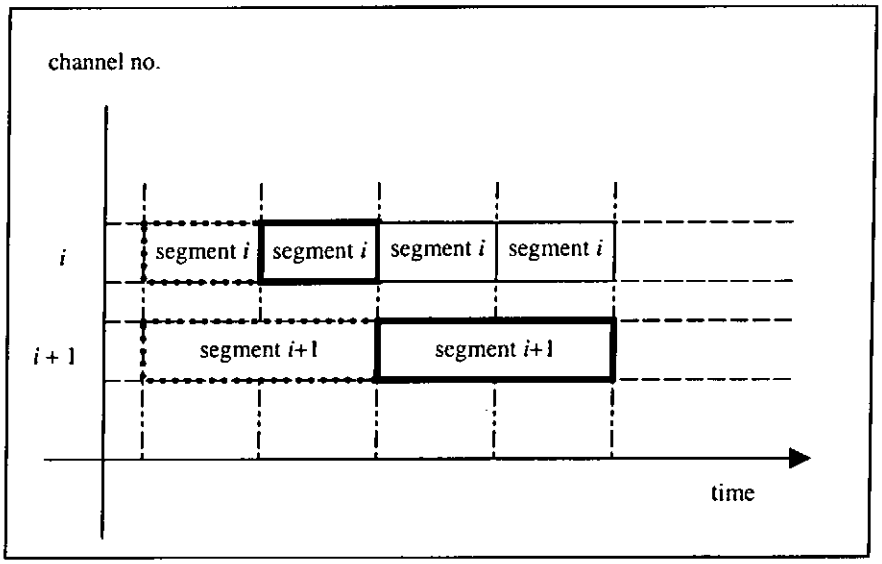

Figure 3.3. The relation between the video segments on two consecutive logical channels of FDB.

The addition of channels in FDB will *not* change the characteristics of the broadcast schedule of the existing channels. This will only change the size of video segments. As a result, the

broadcast schedule's properties of the existing channels will extend to schedules with additional channels. This means that the properties of FDB with K channels can also be found in FDB with $(K + 1)$ channels. Therefore, the probability of forming a JITS between S_0 and S_i is $(1/2)^i$. For example, the probability of forming a JITS between S_0 and S_1 is 1/2, which means that for every two downloading pairs of S_0 and S_1 there will be one just-in-time download problem. Table 3.1 shows the probabilities of S_0 forming a JITS with other segments.

Segment number i	Probability of forming a JITS between S_0 and S_i
1	1/2
2	1/4
3	1/8
4	1/16
5	1/32

Table 3.1. The probability of forming a Just-In-Time Sequence in FDB.

Recall that the JITS is measured from the first video segment. If S_0 forms a JITS with the video segment on channel i, then it is also forming a JITS with the video segments on channels 1, 2, ..., $(i-1)$. •

From Proposition 3.1, we know that the first block of each video segment, there is one and only one suitable timeslot to download. If all these blocks are broadcast in the same timeslot, the peak instances of just-in-time downloading occurs. Thus, for FDB with K logical channels, the maximum instances of just-in-time downloading within one timeslot is $(K - 1)$ since the downloading from the first channel is excluded. In other words, $(K - 1)$ extra channels are needed in order to provide skip-forward interaction in FDB. However, these extra channels can be shared with another video with the same segmentation. This is due to Proposition 3.3, where the highest probability of forming a JITS is 1/2. The unoccupied timeslots in the extra channels can then be used by another video. Therefore, the extra bandwidth required can be reduced by half, that is, $(K - 1)*b/2$ where b is the video playout rate. Table 3.2 shows the needed resource increment if skip-forward interaction is to be supported in FDB.

Original number of logical channels	Extra number of logical channels needed	Logical channels increment (%)	Bandwidth increment (%)
2	1	50	25
3	2	67	33
4	3	75	38
5	4	80	40
6	5	83	42
7	6	86	43

Table 3.2. The extra logical channels and bandwidth required by FDB to support skip-forward.

3.2 Skip-Forward Broadcasting Scheme

Skip-forward interaction repositions the point-of-play to a point within the succeeding block. To perform the action successfully, the succeeding block needs to be already prefetched in the buffer or is being downloaded. Based on this observation, the **Skip-Forward Broadcasting** (SFB) scheme is designed. Details of the scheme will be described in the following subsections.

3.2.1 Channel Design

Suppose that the video server allocates B Mbits/sec network bandwidth to a video. The network bandwidth is divided into K ($= \lfloor B/b \rfloor$) logical channels of b Mbits/sec each and b is the normal playback rate of a video. Each logical channel repeatedly broadcasts a distinct video segment. The logical channels are numbered beginning with zero.

3.2.2 Video Segmentation and Allocation

In SFB, the video file is equally divided into N video segment blocks, where

$$N = 2^{K-1}$$

The total number of blocks is doubled when the number of logical channels is increased by one. Let us denote the jth block of the video as X_j ($j > 0$). The concatenation of all blocks, in the order of increasing block numbers, constitutes the whole video.

$$\text{video file} = X_1 \mid X_2 \mid X_3 \mid \ldots \mid X_{N-1} \mid X_N$$

Figure 3.4 shows two examples of the segmentation of the same video in SFB. From the figure, we can see that there is a regular relationship between the blocks of the two schemes. For instance, the second block of SFB with four channels, denoted as X^4_2, is the concatenation of the two consecutive blocks, X^5_3 and X^5_4, of SFB with five channels. In general, we have the relationship between the blocks under two different segmentations in SFB as follows,

$$X^K_j = X^{K+1}_{2j-1} \mid X^{K+1}_{2j}$$

Figure 3.4. The segmentation of a video in SFB with 4 channels and 5-channels.

Instead of allocating the video segment blocks in a series of geometrically increasing number [1, 2^1, 2^2,...] on the channels as in FDB, the allocation series of SFB with K channels is generated by the following function

$$C(i) = \begin{cases} 1 & i = 0, \\ 1 & i = 1, \\ 2 * C(i-1) & 2 \le i \le (K-1) \end{cases}$$

where $C(i)$ is number of blocks on channel i. The first two channels both have one block, which is X_1 and X_2 respectively. For the other channels, each has the number of blocks two times of that on the previous channel. We shall refer to these contiguous blocks on channel i as video segment i, and denoted as S_i. In general, we have

$$S_i = \begin{cases} X_1 & i = 0 \\ X_{2^{i-1}+1}, X_{2^{i-1}+2}, \ldots, X_{2^i} & i > 0 \end{cases}$$

For SFB with K logical channels, the total number of blocks is:

$$1 + 1 + 2 + 2^2 + \ldots + 2^{K-2} = 1 + (2^{K-1} - 1) = 2^{K-1}$$

If the video segment on the first channel is not taken into account, the allocation series of SFB is the same as in FDB. An example of SFB with four logical channels is shown in Figure 3.5.

video file	1	2	3	4	5	6	7	8

Figure 3.5a. The segmentation of a video in SFB (4 logical channels).

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8

Figure 3.5b. The broadcast schedule of SFB (4 logical channels).

3.2.3 Download Sequence in SFB

New clients have to wait for the broadcasting of the next instance of the first block X_1 of the target video. Once starting the download, X_1 is consumed instantly. The second block X_2 needs to be downloaded from channel 1 at the same timeslot of the downloading of X_1, otherwise skip-forward interaction cannot be guaranteed in the scheme. Subsequent blocks can be downloaded either in a greedy way or in an optimistic way, as illustrated in Figure 3.6. In the greedy way, they are downloaded at their first occurrence whenever the downloading of the first block starts. In the optimistic way, a block is downloaded only if missing of it will result in viewing discontinuity which can be indicated by the playtime difference with respect to the current point-of-play. If the difference is smaller than or equal to the recurring period of the block, then this block has to be downloaded.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8

Figure 3.6a. A greedy download sequence in SFB.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8

Figure 3.6b. An optimistic download sequence in SFB.

Except in the first block, just-in-time downloading is not found in other blocks because the blocks are prefetched before their playtime. Hence, there is no JITS in SFB.

3.3 Performance Analysis

SFB guarantees clients to perform a skip-forward interaction successfully within a range of one block during the whole video duration. The extra attempts depend on the video data in the client's buffer or the download sequence. The next subsection proves the correctness of the guarantee.

3.3.1 Correctness

Case 1. Skip-forward is performed during the consumption of X_1.

The skip-forward interaction will reposition the new point-of-play within the succeeding video segment block X_2. In the download sequence, X_1 and X_2 are downloaded simultaneously. Therefore, part of X_2 is already prefetched in the client's buffer and skip-forward can be carried out. The later skip-forward is performed, the longer the range of new point-of-play selection can be.

Case 2. Skip-forward is performed during the consumption of blocks other than X_1.

In SFB, the download sequence excluding the downloading of X_1 is the same as in FDB but starts with X_2 instead of X_1. Recall that the blocks in FDB are always readahead or just-in-time downloaded. Since X_2 is prefetched one timeslot before its normal playtime, all the subsequent blocks are consequently being prefetched too. This shifting effect guarantees that the succeeding block with reference to the current point-of-play will always be prefetched in the buffer. Thus, skip-forward can be supported and the feasible range of selection is the playback duration of one block.

The shaded area in Figure 3.7 shows the feasible range of skip-forward in SFB.

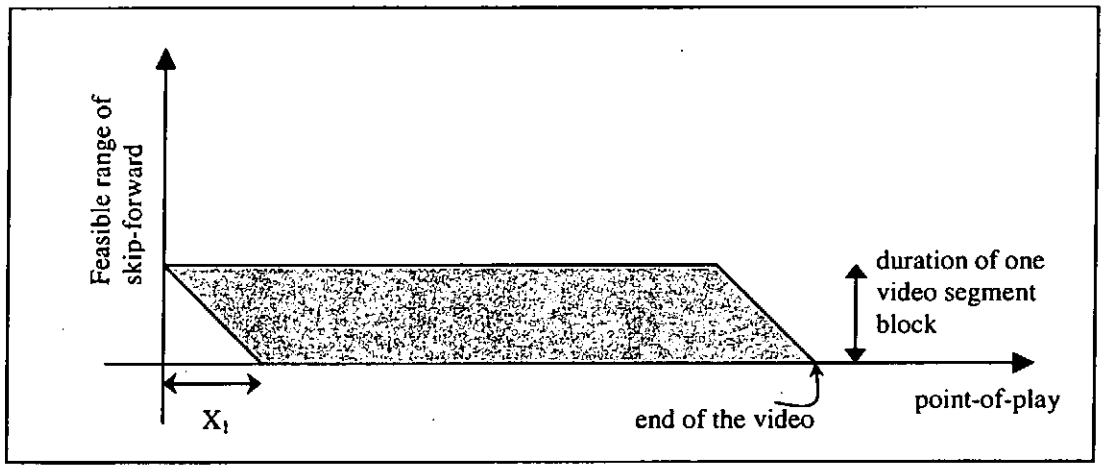

Figure 3.7. Feasible range of skip-forward in SFB.

3.3.2 Startup Latency

Since all the video segments are broadcast at the normal playout rate, we can compute the worst startup latency as the length (i.e., playback duration) of the first video segment, or one block. To broadcast an M-minute video over K channels, we have

$$startup\ latency = (M / 2^{K-1}) \text{ minutes}$$

3.3.3 Client-side Buffer Requirement

To prevent viewing discontinuity, video segment blocks are prefetched in the client-side buffer. The peak buffer size is required when all blocks of the video are downloaded in the shortest time which occurs if blocks on all the channels are downloaded concurrently. Hence, the shortest time needed is equal to the delivery time of 2^{K-2} blocks on the last channel. During the downloading period, the number of blocks consumed is equal to the number of the blocks on the last channel, i.e., 2^{K-2} blocks. Therefore, the client-side buffer requirement, expressed as a fraction of video file size, is

$$Buffer\ size = \frac{2^{K-1} - 2^{K-2}}{2^{K-1}}$$

$$= \begin{cases} 0 & \text{if } K = 1 \\ 1/2 & \text{if } K > 1 \end{cases}$$

The client-side buffer requirement is half the size of the complete video. The download sequence for the peak buffer size required is illustrated in Figure 3.8. In the figure, the table is the broadcast schedule of SFB with four channels. The download sequence starts at *t0*. The chart below the table shows the number of blocks accumulated in the client-side buffer under the corresponding timeslots.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11
channel 0	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8

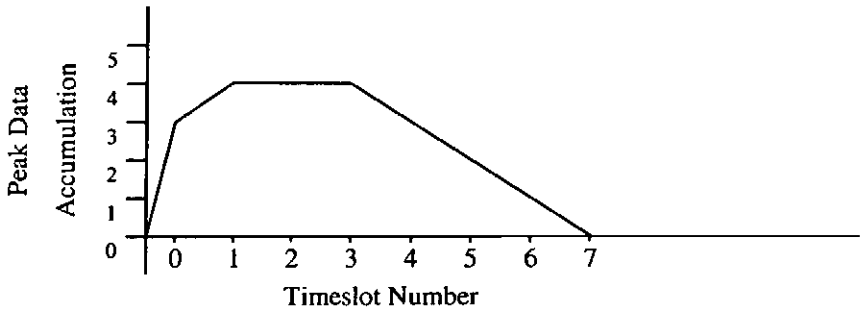

Figure 3.8. The download sequence for the peak client-side buffer size required in SFB (4 logical channels).

3.3.4 Client-side I/O Bandwidth Requirement

At the client side, I/O bandwidth is required for write-to-buffer operation(s). Each instance of downloading from a channel needs one write-to-buffer operation. In SFB with K logical channels, the I/O bandwidth requirement is therefore $(K * b)$, where b is the playout rate of the video.

3.3.5 Comparison

We shall compare the performance between FDB and SFB. The playback duration of the whole video is set to 120 minutes.

A) Startup Latency

Figure 3.9 shows the startup latencies of FDB (without skip-forward support) and SFB. FDB has a shorter startup latency than SFB under the same number of logical channels. It is because SFB used one more channel than FDB in order to support skip-forward interaction. With K logical channels, the total number of video segment blocks in FDB is $(2^K - 1)$ while

in SFB, it is 2^{K-2}. The larger the number of video segment blocks, the smaller the size of a block. Since the startup latency is directly proportional to the size of the first block, the startup latency in FDB is shorter than SFB.

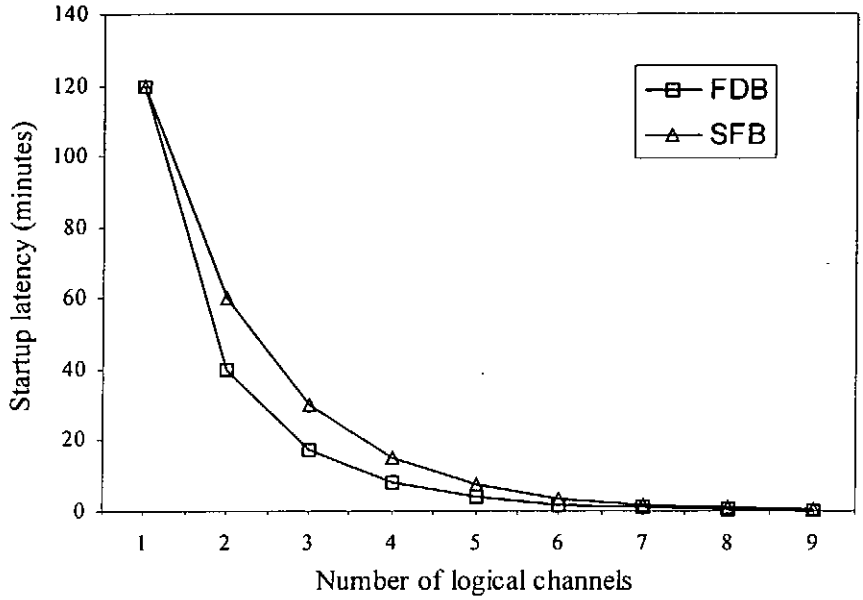

Figure 3.9. The startup latencies of FDB (without skip-forward support) and SFB.

The merit is different if FDB is made to support a skip-forward function. Figure 3.10 shows the startup latencies of SFB and FDB when skip-forward is supported. The startup latency in SFB is now shorter than FDB under the same number of logical channels. This is because FDB needs more than one extra channel to provide skip-forward interaction.

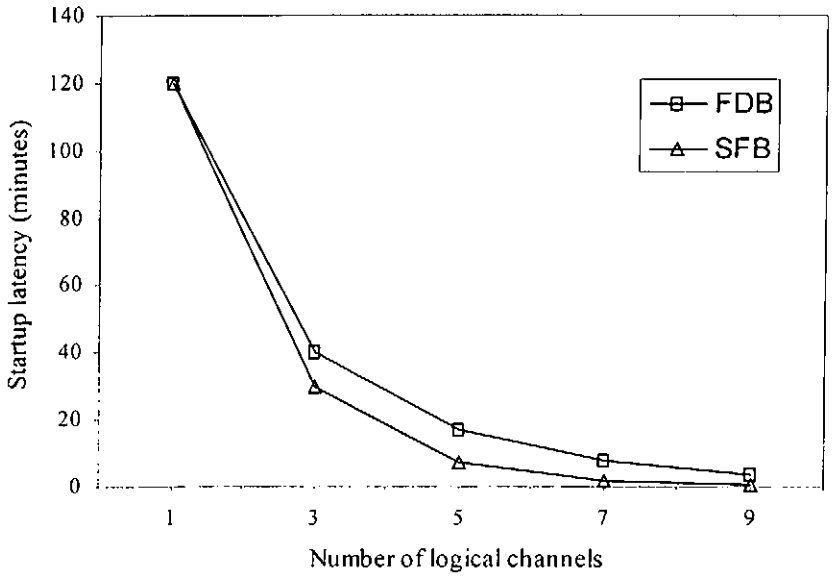

Figure 3.10. The startup latencies of FDB (with skip-forward support) and SFB.

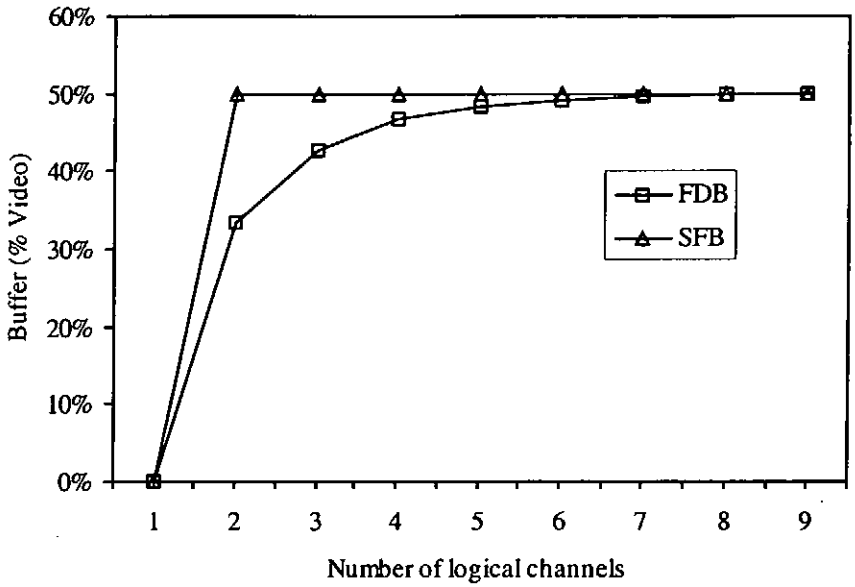

Figure 3.11. The client-side buffer requirements of FDB and SFB.

B) Client-side buffer requirement

Figure 3.11 shows the buffer requirements of FDB and SFB. The buffer size required in SFB stays at half of a video when the number of logical channels is greater than one. In FDB, the buffer requirement converges to half of a video once the number of logical channels is more than four. To achieve startup latency of a few minutes, a video will not be broadcast by less than four channels. Under such condition, the buffer requirement of SFB is comparable to FDB.

3.4 Seamless Channel Transition

Network bandwidth, in terms of the total number of channels available to a VoD system, is a limited system resource. Thus, utilization of the network bandwidth is a key issue to the VoD system performance. In addition, the level of demand on a video, no matter whether it is popular or not, will be affected by many factors, such as holidays and the time of a day. To achieve the best utilization of the network bandwidth, the number of channels allocated to a video should adapt to changes in demand on the video. The adjustment in the number of allocated channels can be an increase (step up) or a decrease (step down). ***Channel transition*** refers to the process by which a broadcasting video undergoes a change on the number of channels allocated to it [31]. Further, it is desirable that such adjustment can be done on-the-fly and seamlessly such that the clients currently viewing the video will not experience any disruption during the channel transitions.

3.4.1 Seamless Channel Transition (SCT) Scheme

As reviewed in Chapter 2, SCT [31] is designed on top of the Fast Data Broadcasting scheme to provide on-the-fly seamless channel transition. The technique used in SCT is called *data padding* where dummy video data is padded at the end of a video.

A) Data Padding

The required size of the dummy video data, V_{dummy}, depends on the base number of channels, α, of the video V. The number α represents the lower bound of the number of channels that should be allocated to V. After a step-down channel transition, the number of channels of V cannot go below α. Assume the playback duration of V is M minutes. Once the number α is chosen, the size of V_{dummy} can be calculated by the following formula:

$$|V_{dummy}| = \frac{M}{2^{\alpha} - 1}$$

After padding, the new video V' is

$$V' = V \mid V_{dummy}$$

In SCT, the padded video is divided into 2^K blocks and the broadcast schedule is the same as that of FDB. Figures 3.12*a* and 3.12*b* show two segmentation examples of SCT with α being 2 and 3 respectively. In each figure, the video is segmented for some K channels.

<table>
<tr><td>V'</td><td colspan="12">V</td><td colspan="4">V_{dummy}</td></tr>
<tr><td>$K = \alpha = 2$</td><td colspan="4">1</td><td colspan="4">2</td><td colspan="4">3</td><td colspan="4">4*</td></tr>
<tr><td>$K = 3$</td><td colspan="2">1</td><td colspan="2">2</td><td colspan="2">3</td><td colspan="2">4</td><td colspan="2">5</td><td colspan="2">6</td><td colspan="2">7^</td><td colspan="2">8*</td></tr>
<tr><td>$K = 4$</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13^</td><td>14^</td><td>15^</td><td>16*</td></tr>
</table>

Figure 3.12a. Data padding in SCT ($\alpha = 2$).

<table>
<tr><td>V'</td><td colspan="12">V</td><td colspan="2">V_{dummy}</td></tr>
<tr><td>$K = \alpha = 3$</td><td colspan="2">1</td><td colspan="2">2</td><td colspan="2">3</td><td colspan="2">4</td><td colspan="2">5</td><td colspan="2">6</td><td colspan="2">7</td><td colspan="2">8*</td></tr>
<tr><td>$K = 4$</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15^</td><td>16*</td></tr>
</table>

Figure 3.12b. Data padding in SCT ($\alpha = 3$).

The dummy video segment blocks are not all to be broadcast in SCT. In the figure, the dummy blocks to be broadcast are marked with "^" behind their block number. Those not to be broadcast are marked with "*".

From the figure, we can observe that

1. The block X_j of the K-channel SCT scheme is equal to the concatenation of the blocks X_{2j-1} and X_{2j} of the $(K + 1)$-channel SCT scheme for any j.
2. The smaller the α, the larger the size of V_{dummy}.
3. With the same α, the more the channels allocated to a video, the larger the portion of V_{dummy} will be broadcast on channels. In other words, more network bandwidth is used, or needed, by the dummy blocks.

The dummy blocks are broadcast periodically on channels. Hence, if we want to reduce the network bandwidth spent on the broadcasting of dummy blocks, a large base number of channels, α, should be chosen. However, a larger α will leave less room for minimizing the bandwidth allocated to a video at run time. Table 3.3 shows the size of dummy video data that will be broadcast on channels for a 120-minute unpadded video under different base number of channels and different allocated channels.

Broadcast dummy video data (minutes)		The number of allocated logical channels					
		2	3	4	5	6	7
Base number of logical channels	2	0	20	30	35	37.5	38.8
	3		0	8.6	12.9	15	16
	4			0	4	6	7
	5				0	1.9	2.9
	6					0	1.4

Table 3.3. The size of broadcast dummy video data under different number of logical channels.

Each row in Table 3.3 is a set of channel allocations with the same α setting. The set is referred to as a ***channel allocation set*** and is denoted as U. The K-channel allocation in the set is denoted as U^K and the channel i of U^K is denoted as U^K_i, where i starts at zero.

B) Channel Transition

In this subsection, we shall provide an in-depth analysis of the step-up and step-down channel transition in SCT. Figure 3.13*a* illustrates the broadcast schedules of SCT ($\alpha = 2$) with two, three and four logical channels. To perform a channel transition, the ending of the first video segments in each schedule have to be aligned together as shown in Figure 3.13*b*. For a better illustration, the blocks in all schedules are marked with the block numbers of the 4-channel SCT scheme and the first video segment in each broadcast schedule is shaded.

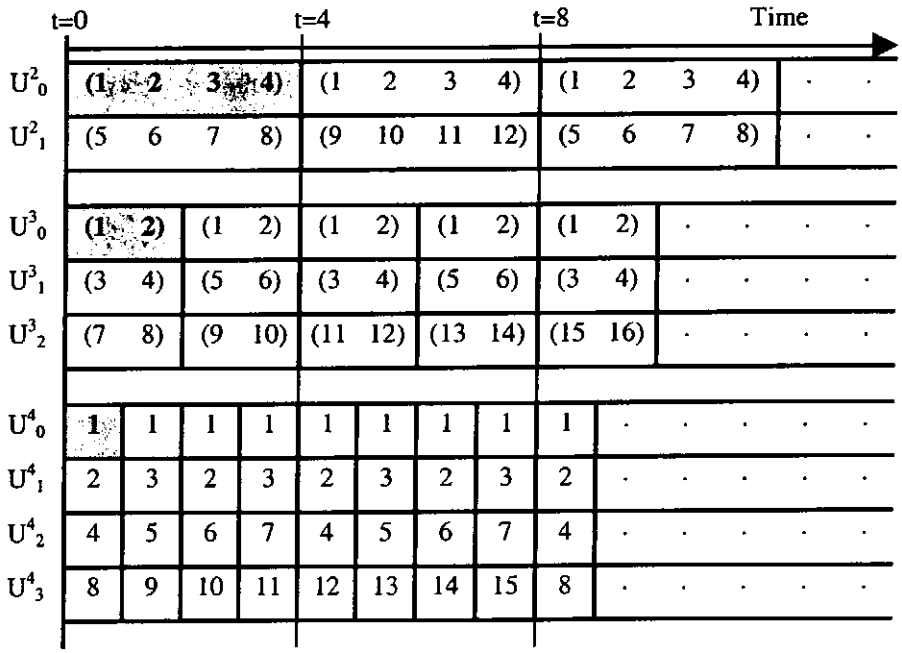

Figure 3.13a. The original broadcast schedules of the 2-, 3- and 4-channel SCT schemes [31].

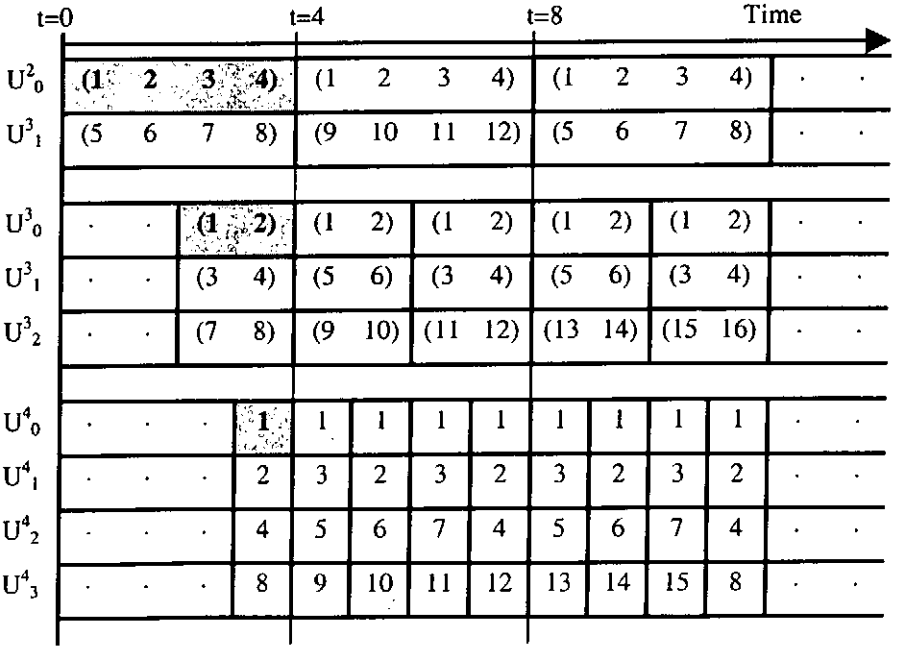

Figure 3.13b. The shifted broadcast schedules of the 2-, 3- and 4-channel SCT schemes [31].

By shifting the timeslots, Tseng [31] proved that there is a relationship between the blocks of U^K and U^{K+1}. Intuitively, the proof states that the block of U^K_i at timeslot j is the concatenation of

1. the blocks of U^{K+1}_i at timeslot $2j$ and U^{K+1}_{i+1} at timeslot $(2j + 1)$, or
2. the blocks of U^{K+1}_{i+1} at timeslot $2j$ and timeslot $(2j + 1)$.

Since the size of a timeslot of U^K is twice that of U^{K+1}, timeslot j of U^K is the same as the concatenation of the timeslot $2j$ and timeslot $(2j + 1)$ of U^{K+1}. As a result, what is broadcast in U^K is also concurrently broadcast in U^{K+1} after the timeslot-shifting. Therefore, in the step-up channel transition, the switching of broadcast schedule from U^K to U^{K+1} could be done seamlessly on-the-fly because the blocks expected by the clients could still be downloaded at the same timeslots.

Unfortunately, in SCT the step-down channel transition is not as straight forward as step-up. In a step-down transition from U^{K+1} to U^K, the number of channels is decreased by one. It implies that at the same timeslot in the broadcast schedule of U^K, *one* block of U^{K+1} will be missed.

Proposition 3.4: In the step-down channel transition from U^{K+1} to U^K, the missing video segment blocks are the last block of the video segments of U^{K+1}. Further, these missing blocks are from the instances of a segment where each left-aligned with an instance of a segment on the next channel.

Proof:
In [31], it is proven that the block of U^K at timeslot j must be the concatenation of two consecutive blocks of U^{K+1}, one with an odd number at timeslot $2j$ and one with an even number at timeslot $(2j + 1)$. In other words, if a block with an odd number at timeslot $2j$ of U^{K+1} cannot find a block with succeeding block number at timeslot $(2j + 1)$, then this block will not be found at the jth timeslot of U^K.

For the broadcast schedules of U, the last block of the segment on channel i is $X_{(2^{i+1}-1)}$, which is an odd-numbered block. Besides, the segment on channel i consists of 2^i (an even

number) blocks. Hence, the last blocks of the segments will be broadcast at even-numbered timeslots. From Proposition 3.3, we know that for every two instances of the segment on channel i, only one of them (which is aligned to the center of an instance of the segment on the next channel) can feasibly form a sequence with an instance of a segment on channel $(i + 1)$. The infeasible one is left-aligned with an instance of succeeding segment such that its last block cannot be followed by a succeeding block. The proof of the proposition is completed. •

Now we know that some blocks will be missed at certain timeslots in the new broadcast schedule after the step-down transition from U^{K+1} to U^K. If these blocks are not made up to the clients, viewing disruption will be experienced. Thus the to-be-returned logical channel has to be tied-up further until all the missing blocks are broadcast.

Proposition 3.5: *In a step-down channel transition from U^{K+1} to U^K, the number of missing blocks needed to be broadcast on the to-be-returned channel is $(K - 1)$.*

Proof:

SCT is inherently a Fast Data Broadcasting scheme with greedy download sequences. Blocks are downloaded at their first occurrence on the channels. Consider the clients who started the viewing one timeslot before the transition. In the download sequence, these clients have to read 2^i blocks from $U^{K+1}{}_i$ for $i = 0$ to K. Thus, after the transition, they are still waiting for $(2^i - 1)$ blocks from $U^{K+1}{}_i$ for $i = 1$ to K. $U^{K+1}{}_0$ is not included because the downloading from this channel is completed. In the worst case, the clients are following a download sequence in which all the last blocks of the segments are missed in the new stepped-down broadcast schedule, i.e., K blocks are missed. On the other hand, by the design property, the last block of the segment on the last logical channel is always a dummy block. The broadcast of the dummy block can be cancelled in the new broadcast schedule. The number of missing blocks, therefore, is $(K - 1)$. •

Since there are time differences among the missing blocks in the broadcast schedule, Tseng suggests to pack the missing blocks together so as to shorten the holding time of the to-be-returned channel. As a result, under a seamless step-down channel transition from U^{K+1} to U^K in SCT, the to-be-returned channel will be released $(K - 1)$ timeslots of U^{K+1} after the starting

of the transition. Table 3.4 shows the holding time of the to-be-returned channels in SCT under different step-down transitions.

Holding time of to-be-returned channels (minutes)		Step-down Channel Transitions				
		From 3 to 2	From 4 to 3	From 5 to 4	From 6 to 5	From 7 to 6
Base number of logical channels of the video	$\alpha = 2$	20	20	15	10	6.3
	$\alpha = 3$		17.1	12.9	8.6	5.4
	$\alpha = 4$			12	8	5
	$\alpha = 5$				7.7	4.8
	$\alpha = 6$					4.8

Table 3.4. The holding time of to-be-returned logical channels under different step-down channel transitions.

3.4.2 Instantaneous Seamless Channel Transition in SFB

In SCT, a video is padded with dummy video data to support channel transition. This dummy video data will be broadcast on the logical channels and will consume valuable server bandwidth. Besides, a video has a base number of channels that at least needs to be allocated. This base number also affects the size of the dummy video data needed in the padding. The larger the base number, the smaller the dummy data size. On the other hand, a large base number will limit the range of step-down channel transitions that can take place. In a step-down channel transition, the to-be-returned channel cannot be released immediately because channel carrying blocks that are missing in the stepped-down broadcast schedule needs to be sustained for some time.

In the Skip-Forward Broadcasting scheme, the dummy video padding, the choosing of a base number of logical channels and the shifting of broadcast schedules are all not necessary to support on-the-fly seamless channel transition. The generic design of SFB already supports this kind of transition. Further, SFB provides *instantaneous* step-down channel transition. That means that the to-be-returned channels can be released immediately upon a downward transition. Before explaining the channel transition in SFB, we shall review the properties of SFB briefly to facilitate the detail analysis.

Properties of SFB:

1. The number of blocks on the first two logical channels (channel 0 and channel 1) is one. Then the size increases doubly as we move up to higher channels.
2. The size of a block in the $(K + 1)$-channel SFB scheme is exactly half of that in the K-channel SFB scheme.
3. The total number of blocks in the $(K + 1)$-channel SFB scheme is twice of that in the K-channel SFB scheme.
4. Because of (2) and (3), we have

$$X^K_j = X^{K+1}_{(2j-1)} \mid X^{K+1}_{2j}$$

where X^K_j denotes the jth block in SFB with m channels.

For an m-channel SFB scheme, let us denote the video segment on channel i as S^m_i, where $0 \le i \le (m-1)$. In general, we have

$$\begin{aligned} S^m_0 &= X^m_1 \\ S^m_i &= X^m_{(2^{i-1}+1)} \mid X^m_{(2^{i-1}+2)} \mid \ldots \mid X^m_{2^i}, \quad \text{for } 1 \le i \le (m-1) \end{aligned}$$

For the segment on channel i of the K-channel SFB scheme where $1 \le i \le (K-1)$, we have

$$\begin{aligned} S^K_i &= X^K_{(2^{i-1}+1)} \mid X^K_{(2^{i-1}+2)} \mid \ldots \mid X^K_{2^i} \\ &= X^{K+1}_{(2(2^{i-1}+1)-1)} \mid X^{K+1}_{(2(2^{i-1}+1))} \mid X^{K+1}_{(2(2^{i-1}+2)-1)} \mid X^{K+1}_{(2(2^{i-1}+2))} \mid \\ &\quad \ldots \mid X^{K+1}_{(2(2^i)-1)} \mid X^{K+1}_{(2(2^i))} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \textit{by the property (4)} \\ &= X^{K+1}_{(2^i+2-1)} \mid X^{K+1}_{(2^i+2)} \mid X^{K+1}_{(2^i+4-1)} \mid X^{K+1}_{(2^i+4)} \mid \\ &\quad \ldots \mid X^{K+1}_{(2^{i+1}-1)} \mid X^{K+1}_{(2^{i+1})} \\ &= X^{K+1}_{(2^i+1)} \mid X^{K+1}_{(2^i+2)} \mid X^{K+1}_{(2^i+3)} \mid X^{K+1}_{(2^i+4)} \mid \ldots \mid X^{K+1}_{(2^{i+1}-1)} \mid X^{K+1}_{(2^{i+1})} \\ &= S^{K+1}_{i+1} \end{aligned}$$

The above derivation shows that S^K_i has exactly the same blocks as S^{K+1}_{i+1} has, i.e., $S^K_1 = S^{K+1}_2$, $S^K 2 = S^{K+1}_3$, $S^K_3 = S^{K+1}_4$, ... , $S^K_{K-1} = S^{K+1}_K$.

Now let us consider video segment S^K_0. Again, by property (4), we have

$$X^K_1 = X^{K+1}_1 \mid X^{K+1}_2$$

In the $(K + 1)$-channel SFB scheme, X^{K+1}_1 and X^{K+1}_2 are repeatedly broadcast on one channel each. By property (2), the sizes of X^{K+1}_1 and X^{K+1}_2 are half of X^K_1. For one broadcast of X^K_1, each of $X^{K+1}_1$and X^{K+1}_2 is broadcast twice. Therefore, the instance of X^K_1 at timeslot j is the concatenation of the X^{K+1}_1 at timeslot $2j$ and the X^{K+1}_2 at timeslot $(2j + 1)$.

Next, we shall discuss the instantaneous step-up/step-down seamless channel transition in SFB. Figure 3.14 shows the broadcast schedules of the 4-channel and 5-channel SFB schemes to aid the discussion.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7
channel 0	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8

Figure 3.14a. The broadcast schedule of SFB (4 logical channels).

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8
channel 4	9	10	11	12	13	14	15	16	9	10	11	12	13	14	15	16

Figure 3.14b. The broadcast schedule of SFB (5 logical channels).

A) Step-up channel transition from K channels to (K + 1) channels in SFB

The download sequences of SFB require the clients to download the video segments on the first two logical channels concurrently. Before a transition, the first two segments are downloaded. Thus, the segments concerned are those on the channels other than these two channels. From the preceding derivation, we know that the segment on channel i of the K-channel SFB scheme, where $1 \le i \le (K - 1)$, is the same as that on channel $(i + 1)$ of the $(K + 1)$-channel SFB scheme. This means that, from channel 1 onwards the segments of the

K-channel scheme can be found at the corresponding timeslots in the $(K + 1)$-channel scheme. Therefore, the step-up channel transition can be launched seamlessly and instantaneously.

B) Step-down channel transition from (K + 1) channels to K channels in SFB

It is assumed that the transition is triggered at the beginning of timeslots in the K-channel scheme because these are the entry points of new clients.

Similar to the case of step-up transition, the video segments concerned are those on the channels after channel 1. From the preceding derivation, we know that the segment on channel i of the $(K + 1)$-channel SFB scheme, where $2 \leq i \leq K$, is the same as that on channel $(i - 1)$ of the K-channel SFB scheme. This means the segments on channels starting from channel 2 of the $(K + 1)$-channel scheme can be found at the corresponding timeslots in the K-channel scheme. Thus, the step-down channel transition can be launched seamlessly. Moreover, as the broadcast schedule of the K-channel scheme covers all the blocks needed by the existing clients of the $(K + 1)$-channel scheme, the to-be-returned logical channel can be released immediately after the starting of the transition. The step-down channel transition in SFB, therefore, can be completed instantaneously.

After the step-down transition, the size of a video segment block is changed and doubled. The skip-forward range support to the clients existing before the transition will still apply to the block size of the old broadcast schedule.

3.5 Pause and Resume Interactions

In this section, the third feature of Skip-Forward Broadcasting scheme in supporting pause and resume interactions is discussed.

Pause and resume functions are among the most commonly used operations on VCRs. A client may choose to stop viewing a video at any point in the video to answer a phone call, to get a drink, etc., and then resume viewing from the paused point after a variable and unpredictable duration.

In unicast VoD systems, a client is the sole viewer of a video stream. The pause/resume interaction can be simply handled by pausing the delivery of video data from the server to the client. Once the client issues the resume command, delivery is restored on the same video stream.

In broadcast VoD systems, however, a video stream is shared by a pool of clients. The video data is broadcast in a predetermined schedule. The video server cannot deviate from its scheduled video data retrievals. On the other hand, clients have to stay in track to the schedule in order to ensure viewing continuity. Any missing video data may cause viewing disruption. The recurrence of the missed video data depends very much on the system design. It may range from a few seconds to tens of minutes. To provide a satisfactory VoD service, it is undesirable to let the clients wait a long time before they can resume viewing after a pause. Therefore, even if viewing is paused, a client should try his/her best to continue the downloading of video data. Now the question is, how much video data has to be downloaded in the pause period in order to provide an instantaneous resume without disruption?

For SFB with K channels ($K > 1$), the total number of blocks is 2^{K-1}. The number of blocks on the last channel is the largest and the recurring period of these blocks is 2^{K-2} timeslots. Clients who missed the downloading of one of these blocks have to wait 2^{K-2} timeslots for its next occurrence. In the pause period, the client may miss a downloading of these blocks. Therefore, the playback duration of the video data in a client-side buffer has to be long enough to cover 2^{K-2} timeslots. In other words, the client-side buffer needs to be able to hold 2^{K-2} blocks, which is half of the size of the complete video, because the playback duration of one block is one timeslot. In Section 3.3.3, we show that the client-side buffer requirement of SFB with i channels is half of a video file size. Therefore, SFB supports pause and resume interactions by nature.

For SFB with K channels, pause/resume interaction can be simply implemented by:

1. During a pause period, the client keeps following the download sequence to download video segment blocks up to 2^{K-2}.
2. Once resume command is issued, the consumption of video data in the buffer restarts. At the same time, the downloading of the remaining blocks starts.

3.6 Summary

In this chapter, a new broadcasting scheme called Skip-Forward Broadcasting (SFB) is introduced. In SFB, a video is segmented into small video segment blocks, which are then broadcast on logical channels operating at a normal video playout rate. The underlying principle of supporting a skip-forward function is to let the clients download the first two video segment blocks at the same time in the beginning. By doing so, there is always at least one prefetched block in the client's buffer to support the skip-forward interaction. Thus, SFB allows a client to reposition the point-of-play in the forward direction within a range of one video segment block. More importantly, SFB supports instantaneous pause/resume interaction and instantaneous on-the-fly seamless channel transition.

Chapter 4: Short-Range Fast-Forward Broadcasting Scheme

4.1 Fast-Forward Interaction

Fast-forward (FF) interaction is a common function provided by VCR. It is different from skip-forward, though both move the point-of-play in the forward direction. Skip-forward is an instant discrete jump to the other point-of-play of the video. Fast-forward, on the other hand, is a forward visible scan of the video at a rate higher than the normal playout rate. Figure 4.1 illustrates the concept of a fast-forward interaction.

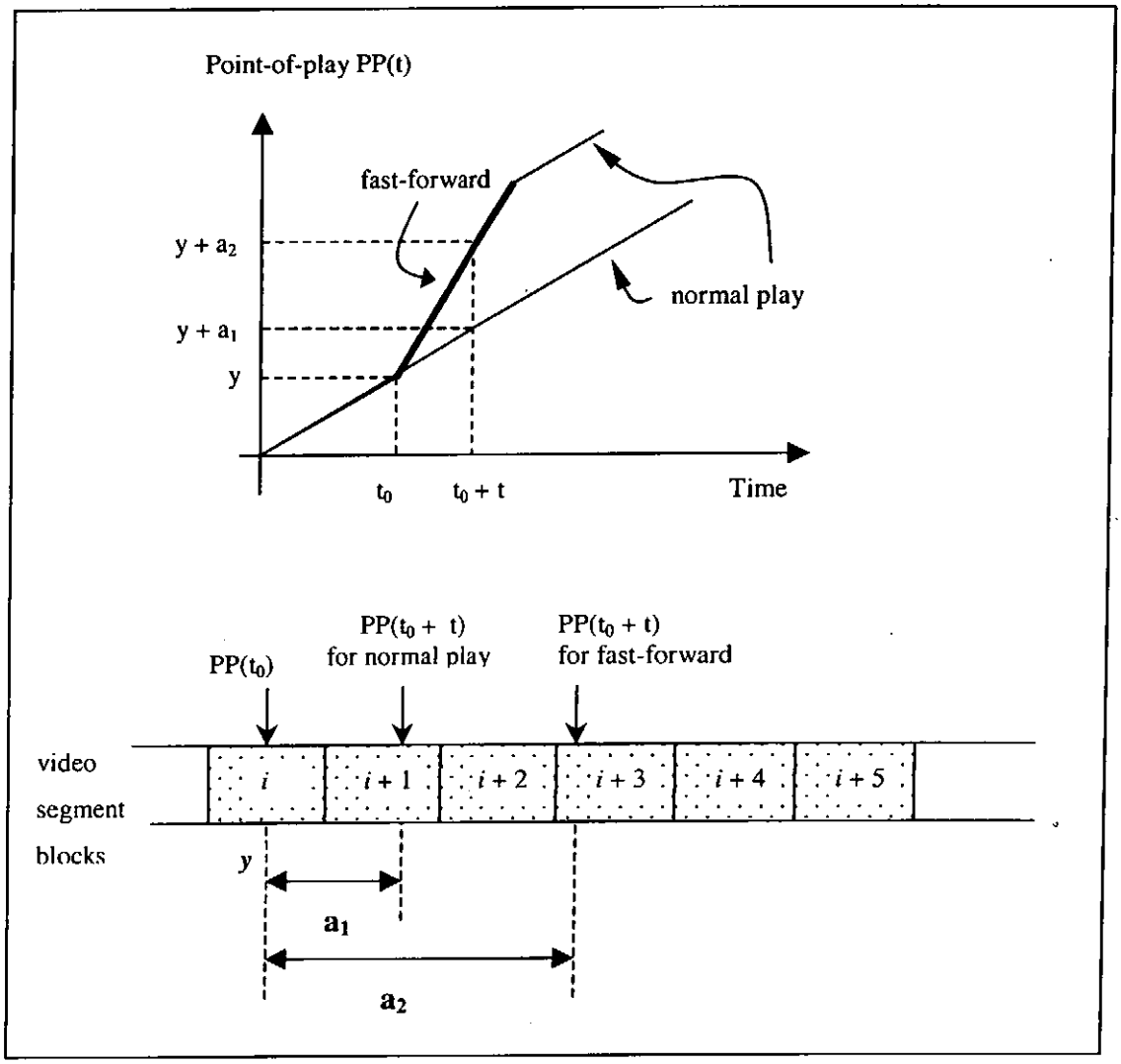

Figure 4.1. Fast-forward interaction.

Let y be the current point-of-play of the video at t_0. After running t units of time, in the normal play mode, the new point-of-play will be moved to $(y + a_1)$. But in the fast-forward

mode, the new point-of-play will become $(y + a_2)$, where $a_2 > a_1$, and the video data from y to $(y + a_2)$ will be consumed within t units of time. The ratio a_2/a_1 is the fast-forward ratio and is greater than one.

Due to the fact that fast-forward interaction is operated at a playout rate higher than the normal rate, a video segment will be consumed in a shorter time than its normal playback duration. Under this situation, buffer may run out of data during the FF period. To cope with this problem, the VoD server can

1. feed the client at the fast-forward playout rate during the interaction, and/or
2. ensure that there are always some up-coming video segments prefetched already in the client-side buffer before the interaction is allowed.

4.1.1 Fast-forward in FDB and SFB

Prefetching is employed in the Fast Data Broadcasting scheme. Video data is often prefetched into the client-side buffer before its playtime. Depending on the download sequence, the prefetched video data need not be the data required in the coming future and thus is not useful to support FF. Besides, FDB has just-in-time downloading problems such that a video segment is consumed while being downloaded. For instance, in half of the download sequences, the second block of the video is a just-in-time downloading. Moreover, in FDB the data transmission rate of channels is equal to the video's normal playout rate. This means that the delivery rate of video data on a channel cannot be increased to the FF rate. Therefore, FDB cannot support fast-forward interactions.

The Skip-Forward Broadcasting scheme provides better support in FF than FDB, though the bandwidth of logical channels in SFB intrinsically supports a normal playout rate only. SFB allows a client to jump forward to another point within the succeeding video segment block. This jumping action requires that the succeeding block has already been prefetched in the client-side buffer or is in the progress of downloading. In SFB, the first two blocks of a video are downloaded simultaneously at the starting timeslot. The subsequent blocks are downloaded afterwards. If no VCR interactions are performed within the starting timeslot, then there will always be at least one successive block kept in the client-side buffer. This implies that an FF could be carried out. But if the FF is requested at the starting timeslot, then it cannot be fulfilled. This is because the first block is in a state of just-in-time

downloading at a delivery rate that cannot be increased. To sum up, SFB supports FF after the consumption of the first block. Although SFB supports FF in most of the timeslots, strictly speaking SFB does not support fast-forward interaction completely.

The incompleteness of fast-forward support in SFB comes from the first block of the video. Once this problem is solved, SFB becomes fast-forward-interaction capable. Based on this observation, a new broadcasting scheme called **Short-Range Fast-Forward Broadcasting** (SRFFB) is proposed. The theory behind SRFFB will be presented in the next section.

4.2 Short-Range Fast-Forward Broadcasting Scheme

As mentioned above, there are two ways to implement fast-forward interaction. One of them is to increase the transmission rate from the VoD server to clients. This technique will be used to solve the deficiency in SFB.

4.2.1 Channel Design

Assuming that the server allocates B Mbits/sec network bandwidth to a video. The bandwidth is then divided into K logical channels, where $(K + 1) = \lfloor B/b \rfloor$ and b is the normal playout rate of the video. The bandwidth of the first logical channel is $2b$ Mbits/sec while the remaining $(K - 1)$ channels are b Mbits/sec. The logical channels are numbered starting at zero.

4.2.2 Video Segmentation and Allocation in SRFFB

SRFFB uses the same method as SFB in segmenting a video. A video file is equally segmented into N video segment blocks, where

$$N = 2^K$$

When the number of channels is increased by one, the total number of blocks is doubled. In other words, the size of a block is halved when the number of channels is increased by one. Let X_j denote the jth block of the video ($j > 0$). The concatenation of all blocks, in the order of increasing block numbers, constitutes the whole video.

$$\text{video file} = X_1 \mid X_2 \mid X_3 \mid \ldots \mid X_{N-1} \mid X_N$$

Figure 4.2 shows two examples of the segmentation of the same video in SRFFB. From the figure, we can see that there is a relationship between the blocks of the schemes. For example, the third block of the 3-channel SRFFB scheme, denoted as X^3_3, is the concatenation of the two consecutive blocks of the 4-channel scheme, namely X^4_5 and X^4_6. In general, the relationship between the blocks under two different segmentations in SRFFB is,

$$X^K_j = X^{K+1}_{2j-1} \mid X^{K+1}_{2j}$$

3-channel	1		2		3		4		5		6		7		8	
4-channel	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Figure 4.2. The segmentations of a video in the 3-channel and 4-channel of SRFFB schemes.

A similar relationship between blocks can also be found in SFB.

The allocation of blocks on the channels of SRFFB, however, is quite different from SFB. The allocation series in SRFFB with K channels is defined by:

$$C(i) = \begin{cases} 2 & i = 0, \\ 2 & i = 1, \\ 2 * C(i-1) & 2 \le i \le (K-1) \end{cases}$$

where $C(i)$ is the number of blocks that made up the video segment on channel i. Let S_i denote the video segment on channel i. On the first two channels, each segment consists of two blocks. On the other channels, the segment's size is two times of that on its previous channel. In general, the blocks of S_i are:

$$S_i = \begin{cases} X_1 + X_2 & i = 0 \\ X_{2^i+1}, X_{2^i+2}, \ldots, X_{2^{i+1}-1}, X_{2^{i+1}} & i > 0 \end{cases}$$

Figure 4.3 shows an example of SRFFB with three logical channels.

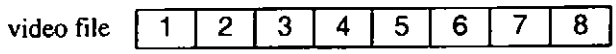

Figure 4.3a. The segmentation of a video in SRFFB (3 logical channels).

timeslot #	t0		t1		t2		t3		t4		t5		t6		t7	
channel 0	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
channel 1	3		4		3		4		3		4		3		4	
channel 2	5		6		7		8		5		6		7		8	

Figure 4.3b. The broadcast schedule of SRFFB (3 logical channels).

4.2.3 Download Sequence in SRFFB

A new client has to catch the next appearance of the first segment S_1 of the target video. Once starting the download, consumption of S_1 starts instantly. Subsequent blocks are downloaded in an optimistic approach or in a greedy approach, as illustrated in Figure 4.4. In the optimistic approach, the blocks are downloaded according to the playtime difference indication. If the playtime difference is smaller than or equal to the recurring period of the block, then this block needs to be downloaded. In the greedy approach, blocks are downloaded at their first occurrence as soon as the downloading of the video begins.

timeslot #	t0		t1		t2		t3		t4		t5		t6		t7	
channel 0	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
channel 1	3		4		3		4		3		4		3		4	
channel 2	5		6		7		8		5		6		7		8	

Figure 4.4a. A greedy download sequence in SRFFB.

timeslot #	t0		t1		t2		t3		t4		t5		t6		t7	
channel 0	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
channel 1	3		4		3		4		3		4		3		4	
channel 2	5		6		7		8		5		6		7		8	

Figure 4.4b. An optimistic download sequence in SRFFB.

4.3 Performance Analysis

The implementation of SRFFB is very similar to SFB. The only difference is that the first two channels of SFB are grouped together to form a single channel with a larger bandwidth in

SRFFB such that the channel can deliver video data at a higher rate. Because of the similarities in the scheme design, the performance of SRFFB is nearly the same as SFB.

SRFFB guarantees that a client can perform one double-rate fast-forward interaction within two blocks at any point-of-play of the video. The extra attempts depend on the client's buffer contents or the download sequence. The correctness of the guarantee is proved below.

4.3.1 Correctness

Case 1. Fast-forward is performed at the beginning of the video.
At the starting timeslot, S_1, which is composed of two blocks X_1 and X_2, is downloaded at a rate twice of the normal playout rate. As a result, the downloading of S_1 is able to keep up with the FF rate for two blocks.

Case 2. Fast-forward is performed during the consumption of the blocks of S_1.
Suppose the FF is performed at time t and denote $|X|$ as the playback duration of one block. Hence, the downloading time of the remaining part of S_1 is $(|X| - t)$. At time t, only $t*b$ video data of S_1 is consumed and the size of non-consumed S_1 is $(2|X| - t)*b$. The playback duration of the non-consumed S_1 in FF mode, which is $(|X| - t/2)$, is longer than the downloading duration of the remaining part of S_1. In the download sequence, the downloading of S_2 will start at the same timeslot as S_1 or just after it. Therefore, during the FF on S_1, the downloading of S_2 has begun. In the worst case, the downloading of S_2 follows that of S_1. At that time, there is still $t*b$ video data of S_1 in the client's buffer. During the consumption of these $t*b$ data, $t*b/2$ data of S_2 will be prefetched in the buffer. Similarly, during the consumption of the $t*b/2$ data of S_2, another $t*b/4$ data S_2 will be prefetched. Since the FF rate is twice of the delivery rate of S_2, the consumption will finally catch up with the downloading of video data such that no more readahead data in the buffer can be used to support FF. Within the FF period, the amount of video data consumed will be $(2|X| - t + t/2 + t/4 + t/8 + t/16 + \ldots)*b$, which tends to be $2*|X|*b$, i.e., the size of two blocks.

Case 3. Fast-forward is performed during the consumption of blocks other than that of S_1.
If no FF is performed during the downloading of S_1, then only the X_1 is consumed and the full X_2 will be prefetched in the client's buffer. Meanwhile, the downloading of the remaining blocks starts concurrently at or after the starting timeslot. This implies that there

will always be at least one block prefetched in the buffer. Thus, the readahead video data in the buffer can at least support an FF period of |X|/2. With the same argument stated in *case 2*, the downloading of video data is still in progress during the FF period. Therefore, the length of the FF period is (|X|/2 + |X|/4 + |X|/8 + ...), i.e., the playback duration of one block. Since the FF rate is twice of the normal playout rate, the video data consumed in the FF period is also two blocks.

Summing up the above three cases, the range of FF supported in SRFFB is two blocks.

4.3.2 Startup Latency

The startup latency is equal to the time between occurrences of the first video segment of the video. Although the first segment is composed of two blocks, the startup latency is still equal to the delivery time of one block because the segment is broadcast at twice of the playout rate. For an M-minute video broadcasting in SRFFB with K logical channels, we have

$$\textit{startup latency} = (M / 2^K) \text{ minutes}$$

4.3.3 Client-side Buffer Requirement

Video segment blocks are prefetched in the client-side buffer to avoid viewing discontinuity. The maximum data accumulation occurs when all blocks of the video are downloaded concurrently in the shortest time. Thus, the shortest time to download all blocks is equal to the time needed to download the last segment, which consists of 2^{K-1} blocks and is the largest among the segments. The number of blocks consumed during the downloading period will also be 2^{K-1}. Therefore, the client-side buffer requirement (in fraction of video file size) is:

$$\begin{aligned} \textit{Buffer size} &= \frac{2^K - 2^{K-1}}{2^K} \\ &= 1/2 \end{aligned}$$

Similar to SFB, the client-side buffer requirement of SRFFB is also half the size of a video.

The download sequence for the peak buffer size is illustrated in Figure 4.5. In the figure, the table is the broadcast schedule of the 3-channel SRFFB scheme. The number of blocks

accumulated in the client-side buffer under the corresponding timeslots is shown in the chart below the table.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11
channel 0	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2	1 2
channel 1	3	4	3	4	3	4	3	4	3	4	3	4
channel 2	5	6	7	8	5	6	7	8	5	6	7	8

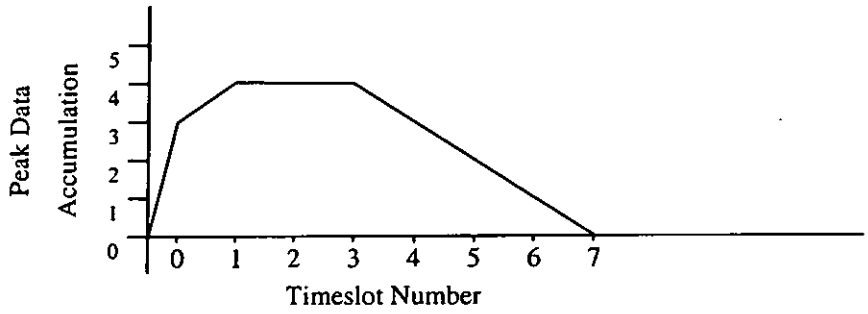

Figure 4.5. The download sequence for the peak client-side buffer size required in SRFFB (3 logical channels).

4.3.4 Client-side I/O Bandwidth Requirement

In the K-channel SRFFB scheme, there are at most K write-to-buffer operations for concurrent downloading from all K logical channels, where one of them is at a rate twice that of the others. The client-side I/O bandwidth requirement, therefore, is $(K + 1) * b$ where b is the video playout rate.

4.3.5 Overall Performance Comparison with SFB

The scheme design of SRFFB is similar to SFB. Therefore, the performance of these two broadcasting schemes is very close. Table 4.1 shows their performance comparison.

	SRFFB	SFB
Number of channels	i	$i + 1$
Network bandwidth	$(i + 1) * b$	$(i + 1) * b$
Client-side buffer requirement (% of video file size)	50%	50%
Client-side I/O bandwidth requirement	$(i + 1) * b$	$(i + 1) * b$
Startup latency for an M-minute video	$M/2^i$	$M/2^i$

Table 4.1. Performance comparison between SRFFB and SFB.

4.4 Seamless Channel Transition in SRFFB

Channel transition deals with the dynamic allocation of bandwidth to a video. It is believed that the network bandwidth should be utilized efficiently because it is limited. The utilization of network bandwidth affects the overall service quality and profitability.

The demand on a video is affected by many factors such as the time of day. If the broadcast schedule of a video can adapt to changes in demand on the video, the VoD system performance can be improved. Additionally, the adjustment should be done on-the-fly and seamlessly such that the existing clients do not experience any viewing disruption when the broadcast schedule is undergoing a transition.

Means such as the dummy video padding, the choosing of a base number of logical channels and the aligning of broadcast schedules (which are used in SCT to support on-the-fly seamless channel transition) are all not needed in SRFFB. The generic design of SRFFB supports the transition already. Unlike SFB, SRFFB cannot provide an *instantaneous* step-down channel transition. This means that the to-be-returned logical channels cannot be released at once upon the step-down transition. Before the discussion of channel transitions in SRFFB, the properties of SRFFB are re-stated below to aid the analysis.

Properties of SRFFB:

1. The number of blocks on the first two channels (channel 0 and channel 1) is two. Then the size increase doubly to the remaining higher channels.
2. The size of a block in the $(K + 1)$-channel SRFFB scheme is exactly half of that in the K-channel scheme.
3. The total number of blocks in the $(K + 1)$-channel SRFFB scheme is twice of that in the K-channel scheme.
4. Because of (2) and (3), we have

$$X^{K}_{j} = X^{K+1}_{(2j-1)} \mid X^{K+1}_{2j}$$

where X^{K}_{j} denotes the block j in the K-channel SRFFB scheme.

Let S^m_i denote the video segment on channel i of the m-channel SRFFB scheme. From Section 4.2.2, we have

$$S^m_0 = X^m_1 \mid X^m_2$$
$$S^m_i = X^m_{2^i+1} \mid X^m_{2^i+2} \mid \ldots \mid X^m_{2^{i+1}}$$

Now let us consider the segment on channel i of the K-channel SRFFB scheme where $1 \le i \le (K-1)$, we have

$$S^K_i = X^K_{2^i+1} \mid X^K_{2^i+2} \mid \ldots \mid X^K_{2^{i+1}}$$

$$= X^{K+1}_{(2(2^i+1)-1)} \mid X^{K+1}_{(2(2^i+1))} \mid X^{K+1}_{(2(2^i+2)-1)} \mid X^{K+1}_{(2(2^i+2))} \mid$$
$$\ldots \mid X^{K+1}_{(2(2^{i+1})-1)} \mid X^{K+1}_{(2(2^{i+1}))} \quad \ldots\ldots\ldots\ldots\ldots \textit{by the property (4)}$$

$$= X^{K+1}_{(2^{i+1}+2-1)} \mid X^{K+1}_{(2^{i+1}+2)} \mid X^{K+1}_{(2^{i+1}+4-1)} \mid X^{K+1}_{(2^{i+1}+4)} \mid$$
$$\ldots \mid X^{K+1}_{(2^{i+2}-1)} \mid X^{K+1}_{(2^{i+2})}$$

$$= X^{K+1}_{(2^{i+1}+1)} \mid X^{K+1}_{(2^{i+1}+2)} \mid X^{K+1}_{(2^{i+1}+3)} \mid X^{K+1}_{(2^{i+1}+4)} \mid$$
$$\ldots \mid X^{K+1}_{(2^{i+2}-1)} \mid X^{K+1}_{(2^{i+2})}$$

$$= S^{K+1}_{i+1}$$

Hence, the blocks of S^K_i is the same as that of S^{K+1}_{i+1}, i.e., $S^K_1 = S^{K+1}_2$, $S^K_2 = S^{K+1}_3$, $S^K_3 = S^{K+1}_4$, ... , and $S^K_{K-1} = S^{K+1}_K$. In addition, these segments are broadcast on channels of the same bandwidth. As a result, the broadcast schedule on channel i of the K-channel SRFFB scheme is equal to that on channel $(i + 1)$ of the $(K + 1)$-channel SRFFB scheme, where $1 \le i \le (K - 1)$. Note that the derivation is in the same principle as in SFB because both have a similar relationship among blocks.

However, the above equality is not applicable to S^K_0. By properties (1) and (4), we have

$$S^K_0 = X^K_1 \mid X^K_2$$
$$= X^{K+1}_1 \mid X^{K+1}_2 \mid X^{K+1}_3 \mid X^{K+1}_4$$
$$= S^{K+1}_0 \mid S^{K+1}_1$$

The segment S^K_0 is split into two segments which are broadcast on two channels of different bandwidth in the $(K + 1)$-channel SRFFB scheme. Thus, equality cannot be applied to these three segments.

Next, we shall discuss the step-up/step-down seamless channel transition in SRFFB. It is presumed that the transition takes place at the end of a timeslot. Figure 4.6 shows the broadcast schedules of the 3-channel and 4-channel SRFFB schemes for illustration. To aid the discussion, the block numbers of the 4-channel scheme is added (with brackets) into the timeslots of the 3-channel scheme.

timeslot #	t0		t1		t2		t3		t4		t5		t6		t7	
channel 0	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)
channel 1	3 (5, 6)		4 (7, 8)		3 (5, 6)		4 (7, 8)		3 (5, 6)		4 (7, 8)		3 (5, 6)		4 (7, 8)	
channel 2	5 (9, 10)		6 (11, 12)		7 (13, 14)		8 (15, 16)		5 (9, 10)		6 (11, 12)		7 (13, 14)		8 (15, 16)	

Figure 4.6a. The broadcast schedule of SRFFB (3 logical channels).

timeslot #	t0		t1		t2		t3		t4		t5		t6		t7		t8		t9		t10		t11		t12		t13		t14		t15	
channel 0	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
channel 1	3		4		3		4		3		4		3		4		3		4		3		4		3		4		3		4	
channel 2	5		6		7		8		5		6		7		8		5		6		7		8		5		6		7		8	
channel 3	9		10		11		12		13		14		15		16		9		10		11		12		13		14		15		16	

Figure 4.6b. The broadcast schedule of SRFFB (4 logical channels).

A) Step-up channel transition from K channels to (K + 1) channels in SRFFB

Before a transition, the downloading of the first segment is completed so that we only need to consider the succeeding video segments. In the above derivation, we showed the equality of the segments on channel i of the K-channel SRFFB scheme and channel $(i + 1)$ of the $(K + 1)$-channel scheme, where $1 \leq i \leq (K - 1)$. As a result, the segment on channels starting from channel 1 of the K-channel scheme can always be found in the equivalent timeslots and channel of the $(K + 1)$-channel scheme. Therefore, the step-up channel transition can be launched seamlessly and instantaneously.

B) Step-down channel transition from (K + 1) channels to K channels in SRFFB

The step-down channel transition in SRFFB is not as smooth as in SFB. Problems exist in the coordination of the three segments S^{K}_{0}, S^{K+1}_{0} and S^{K+1}_{1}. Practically, the transition should be triggered at the beginning of a timeslot in the K-channel scheme which is the entry point of new clients. Equivalently, the transition is triggered at the beginning of an even-numbered timeslot in the $(K + 1)$-channel scheme because the width of a timeslot in the K-channel scheme is two times that in the $(K + 1)$-channel scheme.

Without loss of generality, let us assume that the transition takes place at the beginning of timeslot $(2y + 2)$ of SRFFB with $(K + 1)$ channels. The blocks X^{K+1}_{3} and X^{K+1}_{4} are expected to be downloaded in timeslots $(2y + 2)$ and $(2y + 3)$ respectively. We note that the concatenation of timeslots $(2y + 2)$ and $(2y + 3)$ of the $(K + 1)$-channel scheme equals to the timeslot $(y + 1)$ in the K-channel scheme. After the transition, it is the start of timeslot $(y + 1)$ of the K-channel scheme in which X^{K}_{1} and X^{K}_{2} are broadcast in order. This means that X^{K}_{2}, alias the concatenation of X^{K+1}_{3} and X^{K+1}_{4}, is broadcast at the second half of timeslot $(y + 1)$. The case is illustrated in Figure 4.7.

timeslot #	t(2y)		t(2y + 1)		t(2y + 2)		t(2y + 3)	
channel 0	1	2	1	2	1	2	1	2
channel 1	3		4		3		4	

Figure 4.7a. The broadcast schedules on channel 0 and channel 1 of the $(K + 1)$-channel SRFFB scheme.

timeslot #	t(y)		t(y + 1)	
channel 0	1 (1,2)	2 (3,4)	1 (1,2)	2 (3,4)

Figure 4.7b. The broadcast schedule on channel 0 of the K-channel SRFFB scheme.

As a result, an existing client cannot download X^{K+1}_{3} and X^{K+1}_{4} at the corresponding timeslots of the new stepped-down broadcast schedule. Therefore, the to-be-returned channel has to deliver these two blocks in order to finish the step-down channel transition seamlessly. Table 4.2 compares the holding time of the to-be-returned channel for a 120-minute video under different step-down channel transitions in SRFFB and SCT.

	Holding Time (minutes)	
Channel Transition	SRFFB	SCT
From 7 to 6	1.9	≥ 4.8
From 6 to 5	3.8	≥ 7.7
From 5 to 4	7.5	≥ 12
From 4 to 3	15	≥ 17
From 3 to 2	30	20

Table 4.2. The holding time of the to-be-returned channel under step-down channel transitions in SRFFB and SCT.

After the transition, the size of a video segment block is doubled. The fast-forward interaction support to the clients existing before the transition will still apply to the block size of the old broadcast schedule.

4.5 Pause and Resume Interactions

As explained in Section 3.5, the support of pause and resume interactions in broadcast schemes depends on the client-side buffer size.

For SRFFB with K channels, the number of blocks of a video is 2^K and the largest segment is broadcasting on the last channel. The recurring period of the blocks of the last segment is 2^{K-1}. Clients who missed the downloading of one of these blocks have to wait 2^{K-1} timeslots for its next occurrence. During the pause period, a client may miss the downloading of one of these blocks. Therefore, the playback duration of the video data in a client-side buffer needs to be long enough to cover 2^{K-1} timeslots. In other words, the client-side buffer size has to be at least as large as 2^{K-1} blocks, which is half of a video file size in SRFFB. In Section 4.3.3, we show that the client-side buffer requirement for SRFFB with K channels is half of a video file size. Therefore, SRFFB supports pause and resume interactions.

The implementation of pause/resume interaction in SRFFB is the same as in SFB. Thus, it is skipped here.

4.6 Summary

Fast-forward (FF) operation consumes video data at a rate higher than the normal playout rate. To prevent buffer run out of data, the transmission rate of video data to clients has to be increased. Under this principle, the Short-Range Fast-Forward Broadcasting (SRFFB) scheme for VoD service is proposed in this chapter. In SRFFB, the first two video segment blocks are delivered to clients at a rate twice to the normal playout rate while the others are broadcast at the normal rate. Besides, blocks are prefetched in the client-side buffer. As a result, fast-forward operation can be performed at any point-of-play of the whole video. The FF interaction in SRFFB supports a range of two blocks. Furthermore, SRFFB also supports instantaneous pause/resume interaction and on-the-fly seamless channel transition.

Chapter 5: Mirrored-Pyramid Broadcasting Scheme

5.1 Client-side Buffer Requirement

In the previous chapters, we have discussed several different broadcasting schemes, namely Fast Data Broadcasting (FDB), Skip-Forward Broadcasting (SFB) and Short-Ranged Fast-Forward Broadcasting (SRFFB). These broadcasting schemes can achieve short startup latency under limited network bandwidth. SFB and SRFFB are also designed to support several VCR interactions. However, the client-side buffer requirements in these three schemes are considerably large and approach to half of a video file size. For a 120-minute MPEG-1 video, the buffer required will be 675 Mbytes. This large buffer size will likely dominate the cost of set-top boxes located in the clients' homes.

In this chapter, a broadcasting scheme called **Mirrored-Pyramid Broadcasting** (MPB) scheme will be presented. It will be shown that MPB can lower the client-side buffer requirement to one-third of a video file size.

5.2 Mirrored-Pyramid Broadcasting Scheme

In this section, the implementation and scheme design of MPB will be presented.

5.2.1 Channel Design

With B Mbits/sec network bandwidth allocated to a video, we shall divide the bandwidth into K ($=\left\lfloor \frac{B}{2b} \right\rfloor$) logical channels of $2b$ Mbits/sec each, where b is the normal video playout rate.

To broadcast the video over the K dedicated channels, the video file is partitioned into K video segments each being broadcast on a designated channel. For the ease of segmentation, K only needs to be an odd number.

5.2.2 Video Segmentation and Allocation

Next, we shall decide the allocation series, i.e., how many video segment blocks are to make up the video segment on each channel, and then calculate the total number of video segment

blocks in the scheme. Let us denote the segment on channel i as S_i and the number of blocks of video segment i as $C(i)$. For MPB with K channels, the allocation series is defined by:

$$C(i) = \begin{cases} 1 & i = 0, \\ 2 * C(i-1) & 1 \le i \le p, \\ C(i-1)/2 & p < i \le K-1 \end{cases}$$

where $p = (K-1)/2$ and $K > 2$

$C(i)$ is also a measure of the recurring period of S_i. In the rest of this chapter, we shall call the channels from 0 to $(p-1)$ as the ***leading channels***, the channels from $(p+1)$ to $(K-1)$ as the ***trailing channels*** and channel p as the ***peak channel***.

For any video segment (or equivalently the set of video segment blocks) on the leading channels and the peak channel, the segment is two times larger than the one on its previous channel. However, instead of monotonically doubling the segment size, the segmentation algorithm changes to halving once the segment size reaches a peak value. More precisely, the size of segments on the trailing channels will shrink by a factor of half successively. Thus, the segmentation model of the video is a pyramid with its mirror image, as shown in Figure 5.1.

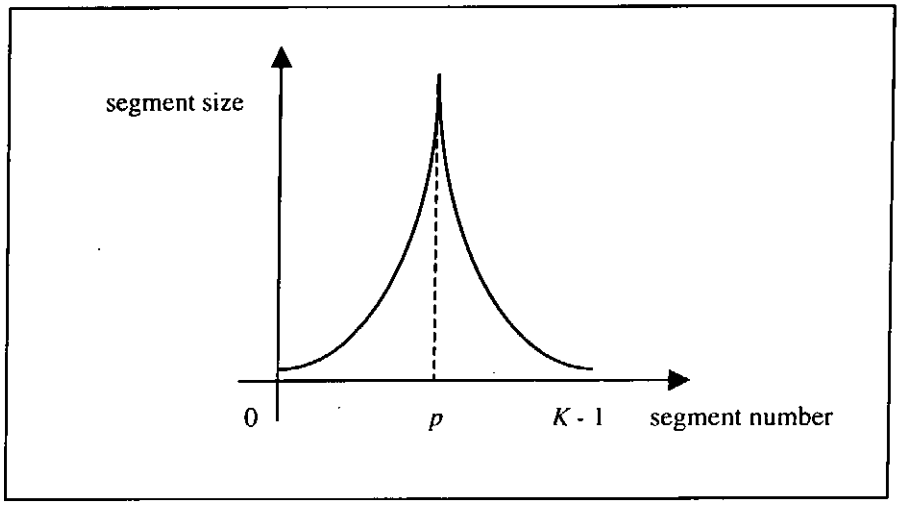

Figure 5.1. The abstraction of segmentation in Mirrored-Pyramid Broadcasting scheme.

Define N be the total number of blocks. From the allocation series, we have

$$\begin{aligned} N &= 1 + 2 + 2^2 + \ldots + 2^{p-1} + 2^p + 2^{p-1} + \ldots + 2^2 + 2 + 1 \\ &= 2 * (1 + 2 + 2^2 + \ldots + 2^{p-1}) + 2^p \\ &= 2 * (2^p - 1) + 2^p \\ &= 3 * 2^p - 2 \end{aligned}$$

Given K channels, the video file has to be equally divided into $(3 * 2^p - 2)$ blocks, where $p = (K - 1) / 2$. The concatenation of all blocks constitutes the whole video. Figure 5.2 shows video segmentation under MPB for 3 channels, 5 channels and 7 channels respectively. From the figure, we can see that there is no regular relationship between the blocks in different segmentations. As a result, channel transition is difficult to be implemented in MPB and is not to be included in the scheme. Figure 5.3 shows the broadcast schedule of the 5-channel MPB scheme.

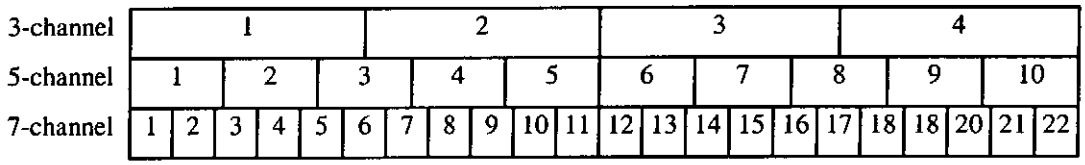

Figure 5.2. The segmentation of a video under MPB for 3 channels, 5 channels and 7 channels.

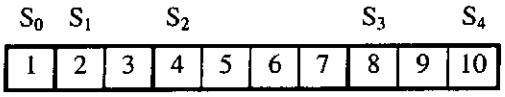

Figure 5.3a. The segmentation of a video in MPB (5 logical channels).

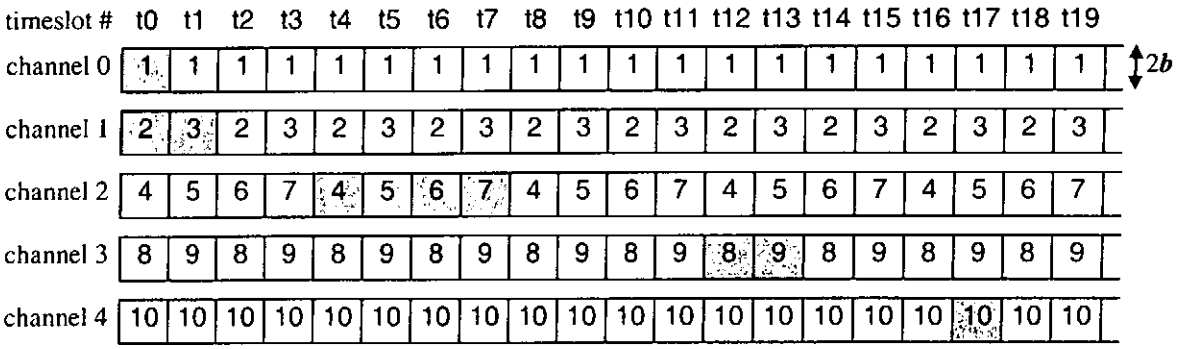

Figure 5.3b. The broadcast schedule of MPB (5 logical channels).

5.2.3 Download Sequence in MPB

In any other broadcasting scheme, new clients have to wait for the occurrence of the next instance of the first segment of the target video. Once downloading of the first segment

begins, playback of the video starts instantly. Download of a subsequent segment appearing on the next channel begins only if missing of it will result in either viewing discontinuity or just-in-time downloading. A download sequence example (the shaded blocks) of MPB is illustrated in Figure 5.3*b*.

Let us recall that the transmission rate in the channels of MPB is twice of the normal playout rate. This means that a block collected in one timeslot can be played for two timeslots. On the other hand, the size of a segment on a channel is at most twice of that on the preceding channel. With these two conditions, we can see that the playback duration of a segment is never less than the recurring period of the segment on the succeeding channel. In other words, the playback duration of a segment can at least cover the time frame occupied by one suitable succeeding segment on the next channel. Therefore, at any point-of-play, the client only needs to download at most two video segments from two consecutive channels concurrently.

5.3 Performance Analysis in MPB

As before, we shall use startup latency, client-side buffer requirement and client-side I/O bandwidth requirement as the performance metrics.

5.3.1 Startup Latency

When a client misses an occurrence of the first video segment S_0 of the requested video, he/she needs to wait for the next occurrence of the segment. To broadcast an M-minute video over $(2p + 1)$ channels, the number of blocks is $(3*2^p - 2)$. A video file size is $(M*b)$ and the size of each block is $(M*b)/ (3*2^p - 2)$, where b is the video playout rate. Since the startup latency is equal to the time to transmit S_0 which has one block only, we have:

$$\begin{aligned} \textit{startup latency} &= \frac{\textit{block size}}{2b} \\ &= \frac{M}{2*(3*2^p - 2)} \end{aligned}$$

5.3.2 Client-side Buffer Requirement

In MPB, clients need to download segments from at most two channels all the time. When the client is downloading segments from channel i and channel $(i + 1)$, the consumption of blocks on channel $(i + 1)$ will not start until the consumption of blocks on channel i completes. Once the consumption of S_{i+1} starts, the downloading from channel $(i + 2)$ will begin. This dual-channel downloading mechanism means that only the blocks from two consecutive channels need to be kept in the client-side buffer at any one time. Therefore, the maximum size among all pairs of consecutive video segments is the client-side buffer requirement.

With $(2p + 1)$ channels, the peak buffer size is required in MPB when the largest pair of segments are downloaded from two channels in the shortest time. Since the segments on channel $(p - 1)$ and channel p are the largest, it is sufficient to consider the buffer required while these two channels are being read. The sizes of video segments on these two channels are 2^{p-1} and 2^p blocks respectively. Within the downloading period of channel p, the number of blocks consumed is equal to that on channel $(p - 1)$. Thus the size of buffer required (in units of blocks) is

$$\begin{aligned} \textit{Buffer size} &= (2^{p-1} + 2^p) - 2^{p-1} \\ &= 2^p \end{aligned}$$

Figure 5.4 shows the buffer requirement in MPB for different number of channels. From the figure, we can see that the buffer requirement in MPB converges to one third of a video file. Practically, a buffer size of 36.4% of the video size will support videos broadcast over four or more channels. The service provider can safely scale up the number of channels for popular videos, without worries on the buffer capacity of the clients' set-top boxes.

5.3.3 Client-side I/O Bandwidth Requirement

The I/O bandwidth requirement at the client side is the total I/O bandwidth required for write-to-buffer operations. In MPB, there are at most two concurrent write-to-buffer operations from two logical channels. Therefore, the client-side I/O bandwidth required is

$$2 * 2b = 4 * b$$

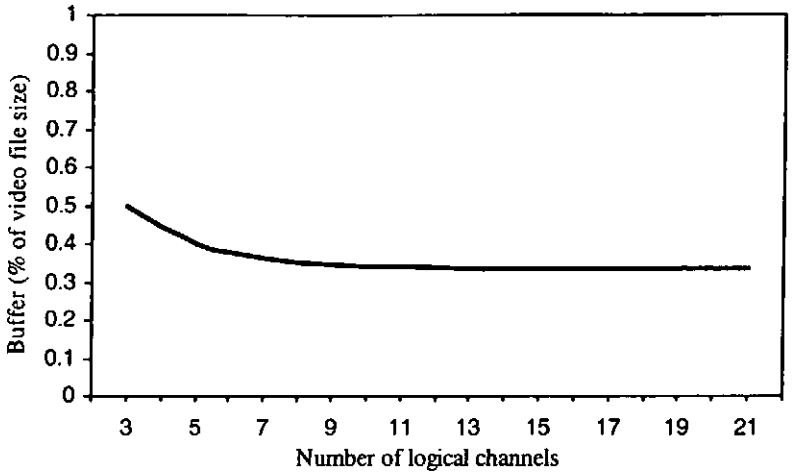

Figure 5.4. The client-side buffer requirement of MPB.

5.4 Scheme Enhancement

Although MPB can reduce the client-side buffer requirement, the network bandwidth required is large. It is because the channels of MPB are operating at twice the video playout rate. In this section, we shall present a technique to reduce the network bandwidth requirement of MPB by packing the segments in the trailing set of channels into one.

5.4.1 Channel-Packing

Recall that from the peak channel onwards, the size of video segments is reduced by half successively. If the size of S_i is 2^j blocks, then the size of S_{i+1} will be 2^{j-1} blocks. In MPB, each video segment is delivered in half of its playback duration. Thus, the time to deliver S_{i+1} is equivalent to the playback duration of 2^{j-2} blocks, which is one quarter of the playback duration of S_i. Next, let us consider the possible starting times of the consumption of S_i.

1. If the starting time of the consumption of S_i is aligned with the beginning of the appearance of S_{i+1} on channel $(i + 1)$, then the consumption period of S_i will cover the time spanned by four instances of integral S_{i+1}.

2. If the starting time of the consumption of S_i is not aligned with S_{i+1}, then the consumption period of S_i will cover the time frame occupied by three integral S_{i+1} and one partial S_{i+1}.

In both of the above cases, starting from the peak channel, the playback duration of *any* video segment will cover *four* instances of suitable succeeding segment on the next logical channel. These cases are illustrated in Figure 5.5*a*. Let us consider the shaded video segment S_2 which consists of blocks 4 to 7 on channel 2. In the best case, consumption of this segment starts at *t4* or *t6* and finishes at *t11* or *t13*. The consumption period of S_2 will cover the instances of S_3 starting at *t4/t6/t8/t10* or at *t6/t8/t10/t12*. In the worst case, consumption will start at *t5* or t7 and will finish at *t12* or *t14*. Again, the consumption period of S_2 will cover four instances of S_3 which start at *t6/t8/t10/t12* or *t8/t10/t12/t14*. In all cases, the number of instances of suitable succeeding S_3 covered is four. As the client does not need choices of more than two suitable instances of S_3 to download, half of the instances of S_3 can be omitted safely. The remaining two suitable instances of S_3 left after the omission will be sufficient. Similar argument can be applied to S_4 while viewing S_3.

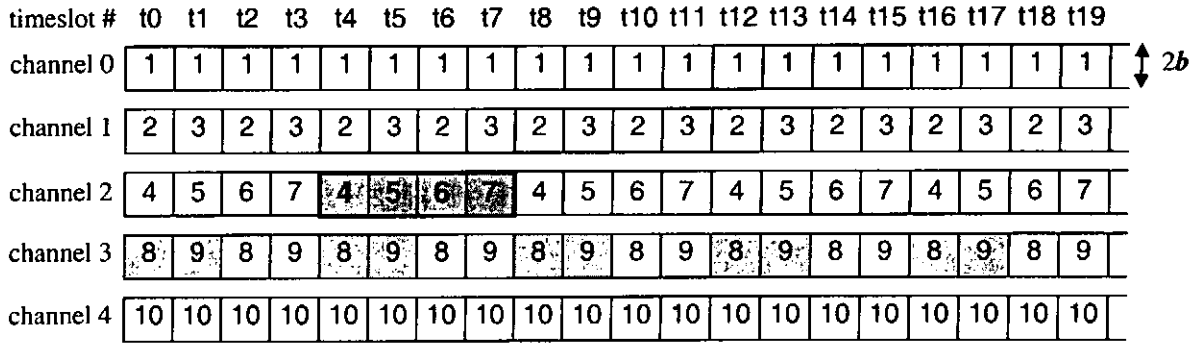

Figure 5.5a. The broadcast schedule of MPB with 10 video segment blocks.

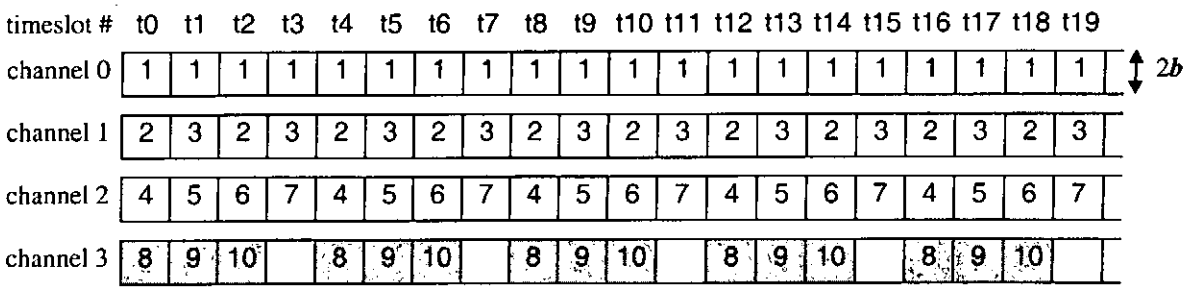

Figure 5.5b. The broadcast schedule of EnMPB with 10 video segment blocks.

With $(2p + 1)$ channels, the peak channel in MPB is channel p. Suppose we now omit half of the instances of segment broadcast on the channel right after the peak channel. As illustrated in Figure 5.5*a*, only the shaded video segments on channel 3 (the channel right after the peak channel) are broadcast. Then, half of the timeslots in channel $(p + 1)$ is freed. The capacity of the contiguous freed timeslots on channel $(p + 1)$ is the same as the size of S_{p+1}, which can be expressed (in units of blocks) as

$$
\begin{aligned}
C(p+1) &= C(p+2) + C(p+2) \\
&= C(p+2) + C(p+3) + C(p+3) \\
&= C(p+2) + C(p+3) + C(p+4) + \ldots + C(2p) + C(2p) \\
&= \sum_{j=p+2}^{2p} C(j) + C(2p) \quad > \quad \sum_{j=p+2}^{2p} C(j)
\end{aligned}
$$

The above shows that the capacity of the freed timeslots is larger than the sum of the size of segments on all channels after channel $(p + 1)$. This allows us to make use of the vacated timeslots in channel $(p + 1)$ to accommodate all the segments on channels after channel $(p + 1)$, as shown in Figure 5.5*b*. The segments on the trailing channels can then be packed together to form a single large contiguous segment of size $(2^p - 1)$ blocks. In other words, all segments on the trailing channels (consists of p channels) are now packed and broadcast on one logical channel, channel $(p + 1)$. After channel-packing, the recurring period of S_{p+1} in the new MPB will be the same as S_p. The new MPB scheme is referred to as **Enhanced MPB** (EnMPB) in the rest of the chapters. The total number of channels required in EnMPB becomes $(2p + 1) - p + 1$, or, $p + 2$.

5.4.2 Client-side Buffer Requirement in EnMPB

Since the broadcast schedule of EnMPB is different from MPB, the client-side buffer requirement in EnMPB may not be the same as MPB. Hence, it is necessary to analyze the buffer requirement in EnMPB.

For EnMPB with $(p + 2)$ channels, the broadcast schedule in the first $(p + 1)$ channels is inherited from MPB. Therefore, the maximum size of buffer required during the downloading from these channels is 2^p blocks, which occurs when the segments on channel $(p - 1)$ and channel p are being downloaded at the same time.

Next, let us consider the size of the buffer required during the downloading from the last two channels, p and $(p + 1)$. The recurring periods of S_{p+1} and S_p are both 2^p timeslots. The consumption period of S_p will always cover the time spanned by two feasible instances of S_{p+1} to be downloaded. To reduce the buffer required, the second, instead of the first, feasible S_{p+1} can be chosen. This implies that the downloading of S_{p+1} is not to be started at the same timeslot when downloading of S_p starts. Suppose that the consumption of S_p starts at time t and ends at $(t + 2*2^p)$. If we have started the downloading of S_{p+1} before $(t + 2^p)$, the

downloading will be finished before $(t + 2*2^p)$. Hence the earliest time to download S_{p+1} shall be $(t + 2^p)$. In fact, between $(t + 2^p)$ and $(t + 2*2^p)$, there will always be another instance of S_{p+1} broadcast on channel $(p + 1)$. Therefore, the client can safely refrain from downloading the first instance of S_{p+1} appearing before $(t + 2^p)$.

Now let us assume that S_{p+1} is downloaded at $(t + 2^p)$. In the worst case, at this timeslot the number of prefetched blocks of S_p in the client-side buffer is 2^{p-1} because:

1. If the number of blocks of S_p in the buffer is more than 2^{p-1}, the playback duration of these blocks will be more than 2^p timeslots. Then the downloading of S_{p+1} will be finished before the blocks of S_p in the buffer can be exhausted. This implies that the S_{p+1} downloaded is not the second feasible instance which contradicts our assumption.

2. If the number of blocks in the buffer is smaller than 2^{p-1}, the blocks of S_p in the buffer will be exhausted before the downloading of S_{p+1} is finished.

Therefore, there are at most an accumulation of 2^{p-1} video segment blocks of S_p at the time downloading of the second feasible instance of S_{p+1} begins.

We now move on to calculate the buffer required during the downloading from channel p and channel $(p + 1)$. The size of S_{p+1} is $(2^p - 1)$ blocks. Before downloading S_{p+1}, there are at most 2^{p-1} readahead blocks of S_p left in the buffer. During the downloading of S_{p+1}, the number of blocks consumed is $(2^p - 1)/2$. As a result, the buffer required (in units of blocks) is

$$\begin{aligned} \textit{Buffer} \text{ size} &= (2^p - 1) + 2^{p-1} - (2^p - 1)/2 \\ &= 2^p - 1 + 2^{p-1} - 2^{p-1} + 1/2 \\ &= 2^p - 1/2 \\ &< 2^p \end{aligned}$$

The buffer required for downloading from channel p and channel $(p + 1)$ is less than 2^p blocks (which is the buffer required for downloading from channel $(p - 1)$ and channel p). Thus, the client-side buffer requirement in EnMPB is also 2^p blocks.

5.5 Performance Comparison

In this section, the results of the analysis on the performance between EnMPB and MPB schemes are presented. Then, the performance of EnMPB, FDB (Fast Data Broadcasting) and SFB (Skip-Forward Broadcasting) are compared. In the analysis, the playback duration of the whole video is set to 120 minutes.

5.5.1 EnMPB and MPB

A) Network Bandwidth Requirement

The first analysis between EnMPB and MPB focuses on the network bandwidth requirements. As shown in Figure 5.6, under the same segmentation approach, the network bandwidth required in EnMPB is smaller than MPB. This is because EnMPB has a better broadcast schedule such that the number of logical channels required is reduced. The saving of channels in EnMPB can be up to about 50%.

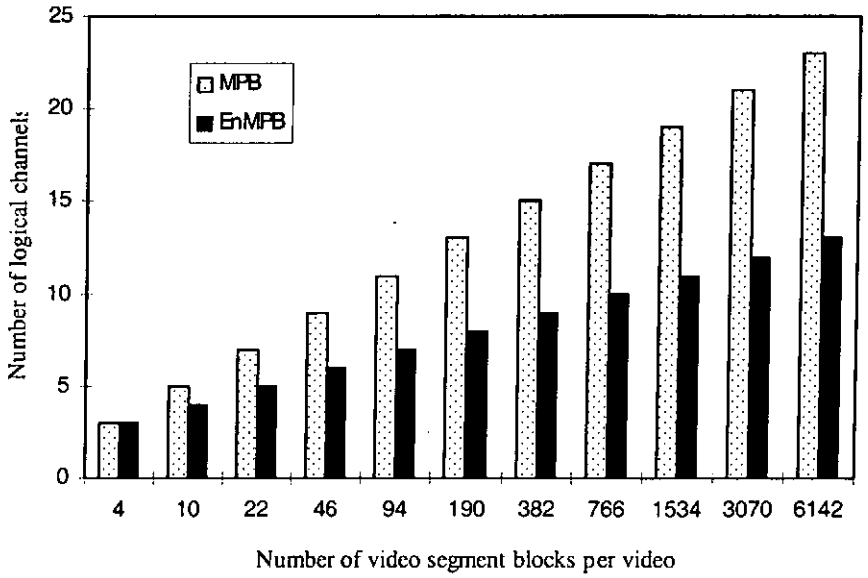

Figure 5.6. The network bandwidth requirements in MPB and EnMPB under the same segmentation.

B) Startup Latency

Figure 5.7 shows the startup latency for both schemes under the same number of channels allocated per video. The startup latency in EnMPB is significantly less than MPB. The reason is that the new broadcast schedule of EnMPB makes a better channel utilization. The number of segments of EnMPB is about twice than that of MPB, making the size of first segment much smaller. As the startup latency is in direct proportion to the size of the first segment, the startup latency in EnMPB is much shorter than MPB.

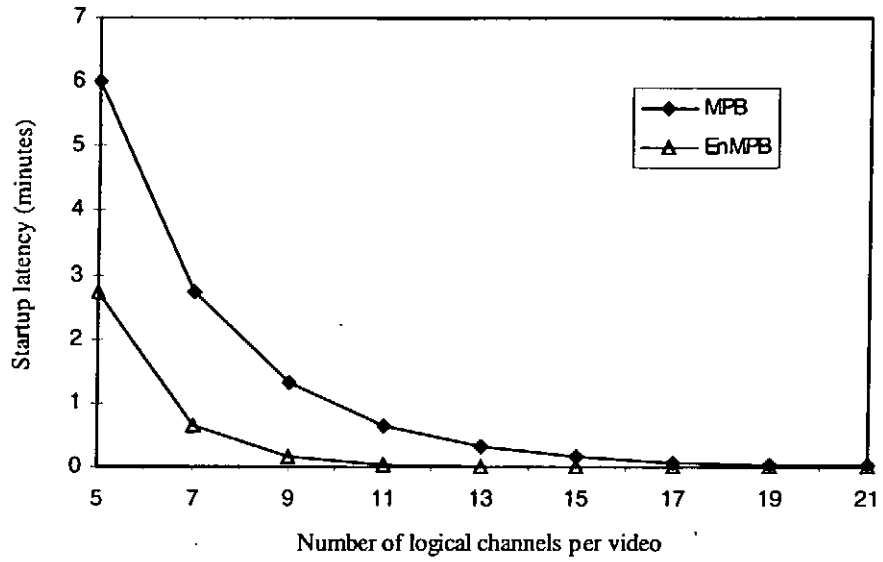

Figure 5.7. The startup latencies in MPB and EnMPB.

C) Client-side Buffer Requirement

The client-side buffer requirements for the schemes with the same number of logical channels allocated per video are shown in Figure 5.8. Again, EnMPB has a better performance than MPB. The buffer requirement is smaller in EnMPB, especially when the number of channels per video needs to be kept small.

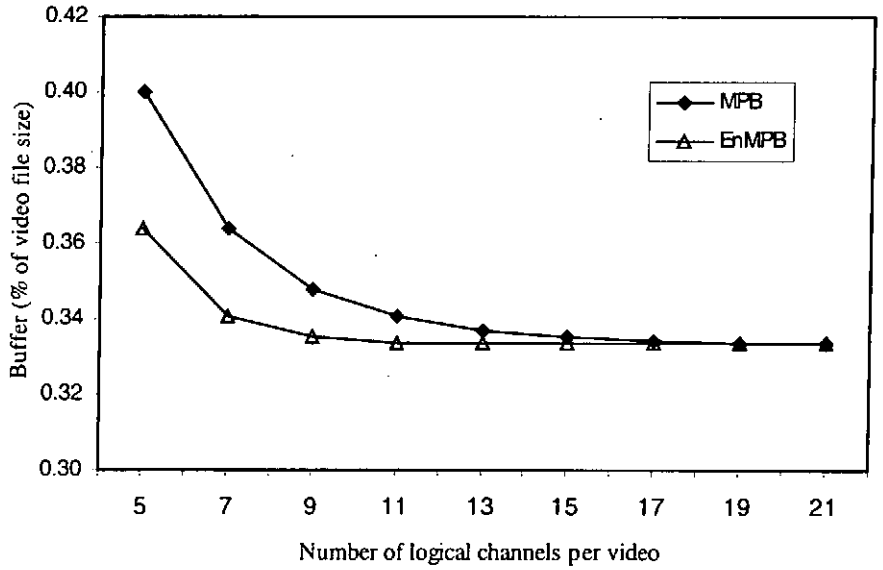

Figure 5.8. The client-side buffer requirements in MPB and EnMPB.

5.5.2 EnMPB, FDB and SFB

Client-side buffer requirement, client-side I/O bandwidth requirement and startup latency are employed as the performance metrics for comparison among EnMPB, FDB and SFB.

A) Client-side Buffer Requirement

Figure 5.9 shows the client-side buffer requirements under different network bandwidth per video. It can be observed that EnMPB requires the smallest buffer size. As the bandwidth allocated to a video increases, the buffer requirement in FDB and SFB are always 50% of a video file size while EnMPB drops to 34% quickly.

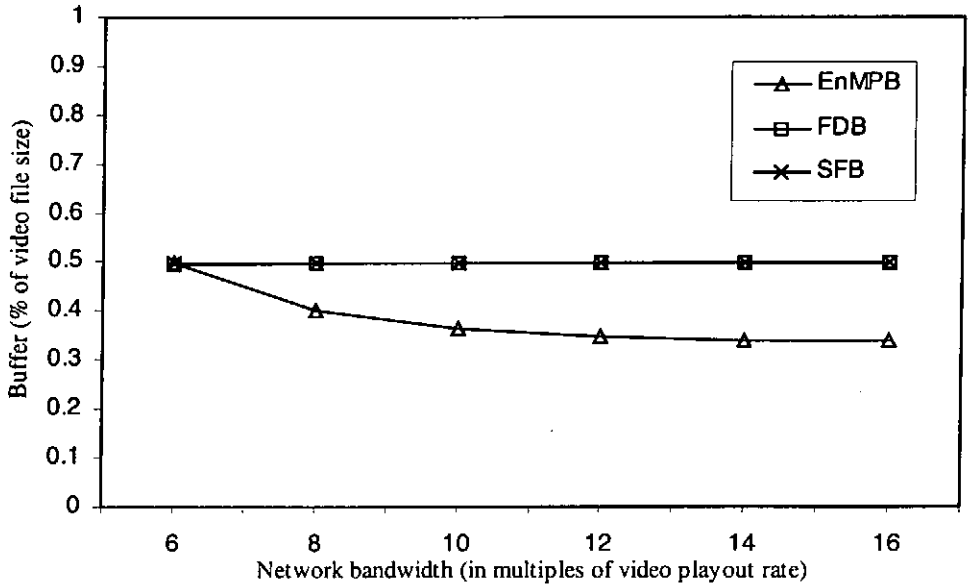

Figure 5.9. The client-side buffer requirements in EnMPB, FDB and SFB.

B) Client-side I/O Bandwidth Requirement

The client-side I/O bandwidth requirements for the schemes are shown in Figure 5.10. It can be seen that the I/O bandwidth requirements in FDB and SFB grow linearly with the allocated network bandwidth. It is because both schemes need to download video data from all logical channels concurrently in order to provide a continuous viewing service. The bandwidth requirements in FDB and SFB, therefore, are unbounded. More importantly, EnMPB does not need more I/O bandwidth regardless of changes in the network bandwidth allocated for a video. That is, the download capability of the installed set-top boxes will not hinder the service provider from increasing the bandwidth allocated to a video. The I/O bandwidth in MPB is the least among the three schemes.

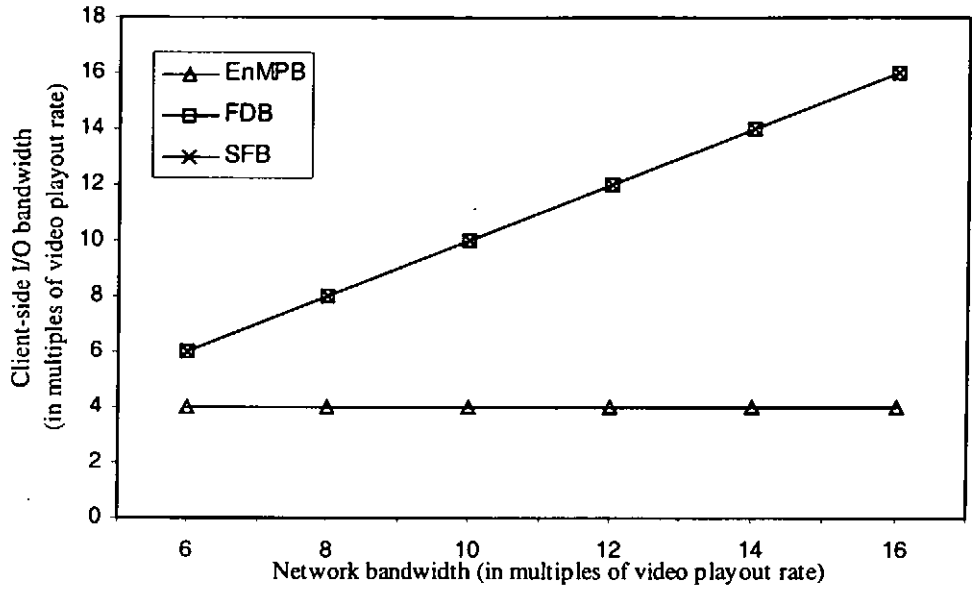

Figure 5.10. The client-side I/O bandwidth requirements in EnMPB, FDB and SFB

C) Startup Latency

The performance in startup latency for the three schemes is shown in Figure 5.11. We can see that the startup latency of EnMPB is the largest among the schemes. It is because the bandwidth per channel in EnMPB is twice that of FDB and SFB. With the same network bandwidth, the segmentation in EnMPB is coarser than the others. Since the startup latency is directly proportional to the size of the first video segment, the startup latency in EnMPB is inevitably larger than the other two schemes.

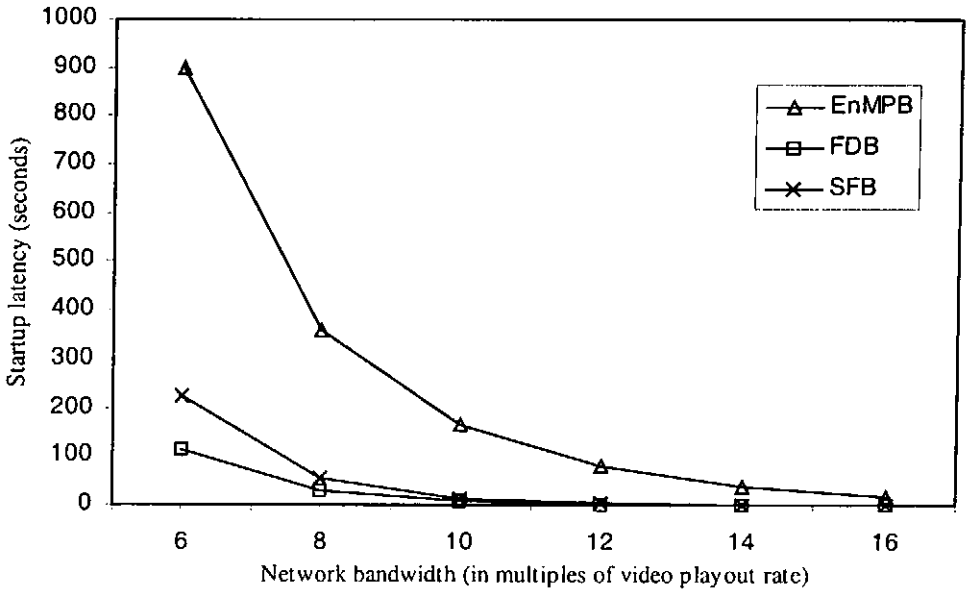

Figure 5.11. The startup latencies in EnMPB, FDB and SFB.

When the network bandwidth allocated to a video is increased to eight times the video playout rate, the startup latencies of all schemes will all be brought down to the order of minutes. which should well be acceptable to home video consumers. Therefore, the absolute differences in startup latency among schemes become much smaller once a reasonable network bandwidth is allocated.

5.6 Fast-Forward Interaction in EnMPB

In EnMPB, the video data is delivered at twice of the normal video playout rate. Thus, by nature EnMPB should be able to support double-rate fast-forward interaction. Although all video segments are delivered at a rate higher than normal, the scheme is still unable to support fast-forwarding throughout the whole video. Let us take a look at Figure 5.12.

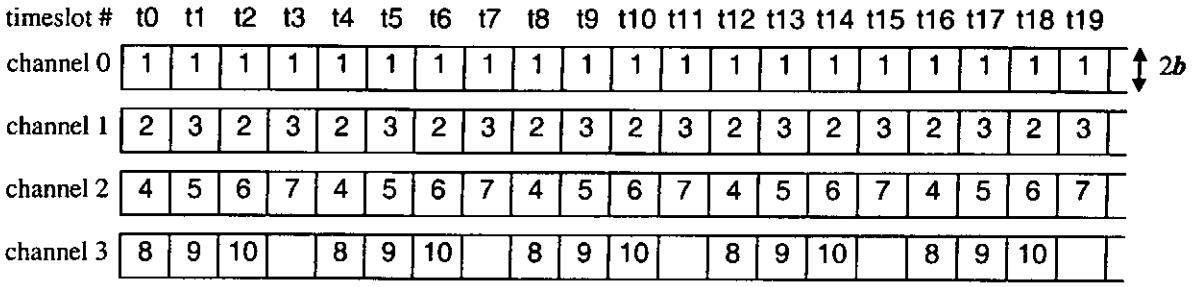

Figure 5.12. The broadcast schedule of EnMPB (4 logical channels).

Figure 5.12 is an example of EnMPB broadcast schedule. If all video segment blocks are consumed at the double playout rate, EnMPB will behave similarly to FDB in which a block can be playback for one timeslot. Recall that in FDB, downloading video segments from all logical channels concurrently is needed in order to ensure viewing continuity. In EnMPB, only two logical channels are monitored at any one time. Thus, during the FF period, the buffer may run out of data before refilling.

Next, we shall derive the range within which double-rate fast-forward is supported by EnMPB. Note that the just-in-time downloading restriction is relaxed during the FF period.

Case 1: FF is performed during the consumption of the video segment on the last channel

In any download sequence, a video segment is downloaded in whole every time. Thus before the start of FF interaction, the last video segment is either already prefetched in the buffer or being downloaded. In both scenarios, FF can be performed within the whole last video segment continuously.

Case 2: FF is performed during the consumption of the video segment on the second-last channel

The video segments on the last two channels have the same recurring period. A new instance of the last video segment will be broadcast on the last channel during the FF period or just after the FF period of the second-last video segment. In both scenarios, the buffer will not run out of data. Thus, FF can be performed within the last two video segments continuously.

Case 3: FF is performed during the consumption of the video segment on a channel other than the last two channels.

Suppose the downloading of the video segment, say S_i, starts at timeslot t. Due to the fact that just-in-time downloading is not allowed in the download sequence during the normal playback, the downloading of a video segment must start before the consumption of the segment. Hence, in the worst case, the consumption of S_i may start at one timeslot just after the starting time of its downloading, i.e., at timeslot $(t + 1)$. Since the size of S_i is 2^i blocks, the consumption of S_i will finish at timeslot $(t + 1 + 2*2^i)$. Applying the same principle, the downloading of S_{i+1} may start at timeslot $(t + 2*2^i)$.

Now let us assume the FF period on S_i lasts for y timeslots. In order to prevent viewing discontinuity, the playback (including FF period) of S_i should not be allowed to be finished before $(t + 2*2^i)$. Thus, we have

$$\begin{aligned} (t+1) + y + 2*(2^i - y) &\geq t + 2*2^i \\ 1 + y - 2y &\geq 0 \\ 1 &\geq y \\ y &\leq 1 \end{aligned}$$

In this worst case, the FF range on S_i can only last for one timeslot, or the client can only play one block in FF mode.

After the consumption (included FF) of S_i, video segment S_{i+1} starts to be downloaded and be playback instantly. The downloading of S_{i+2} will start either at the same time with S_{i+1} or just after the downloading of S_{i+1}. Since the video data is always delivered at twice of the playout rate, the FF interaction can be performed again. For instance, the client performs FF on S_{i+1} immediately after the consumption of S_i completes. Let us assume that the downloading of S_{i+1} starts at timeslot t_1. The downloading of S_{i+2}, in the worst case, starts at timeslot $(t_1 + 2^{i+1})$. Supposing that the FF on S_{i+1} lasts for y_1 timeslots this time, we have

$$\begin{aligned} t_1 + 2*(2^{i+1} - y_1) + y_1 &\geq t_1 + 2^{i+1} \\ 2*2^{i+1} - 2y_1 + y_1 &\geq 2^{i+1} \\ 2^{i+1} - y_1 &\geq 0 \\ y_1 &\leq 2^{i+1} \end{aligned}$$

The feasible FF range in the second attempt is 2^{i+1} timeslots which is much longer than that in the first attempt and is equal to the delivery time of S_{i+1}. This means that the FF interaction immediately after the first attempt can be performed within the whole S_{i+1} video segment.

Summing up the three cases, the clients can always perform double-rate FF for one block. In EnMPB, the FF interaction can be performed more than once easily in all the download sequences. In SRFFB, however, the second and subsequent attempts depend on buffer content and are unlikely to be successful because the video data is delivered only at the normal playout rate.

5.7 Summary

The Mirrored-Pyramid Broadcasting (MPB) scheme can reduce the client-side buffer requirement down to one-third of a video file size. As the video data is broadcast at a rate twice of the playout rate, clients only need to download video data from no more than two logical channels simultaneously at any time. Thus, the I/O bandwidth and the buffer requirements in the set-top boxes are both minimized. Furthermore, an enhancement to MPB is also given to reduce the network bandwidth requirement to around half of the original without affecting the other performance metrics. The enhanced MPB scheme also supports fast-forward interaction.

Chapter 6: On Minimizing Client-side Buffer in SFB and SRFFB

6.1 Introduction

In the previous chapter, a new broadcasting scheme called Enhanced Mirrored-Pyramid Broadcasting (EnMPB) which can reduce the client-side buffer requirement to one third of a video file size is presented. However, unlike the other two broadcasting schemes, Skip-Forward Broadcasting (SFB) and Short-Range Fast-Forward Broadcasting (SRFFB), EnMPB does not support channel transition. This means that EnMPB would not adjust the network bandwidth allocation so as to adapt to the changes in the demand of the videos. On the other hand, both SFB and SRFFB apparently require a large client-side buffer size of half of a video file. Therefore, if we can reduce the buffer requirements in SFB and SRFFB, then these two schemes will be more practical and competitive.

Next, a technique to minimize the client-side buffer requirement for optimistic downloading sequences in SFB will be discussed. The same technique can be applied to SRFFB and the results should be similar.

6.2 Small-Buffer Skip-Forward Broadcasting (SB-SFB) Scheme

Recall that in SFB, video segment blocks are prefetched into the client-side buffer to avoid viewing discontinuity. The maximum data accumulation occurs when the blocks are all prefetched in the shortest time. This is because the buffer required is computed as the difference between the total number of blocks of a video and the number of blocks consumed during the whole downloading period. Therefore, if we can elongate the shortest download sequence length, or shortest-length in brief, to download all the blocks, the number of blocks consumed during the lengthened downloading period will increase and hence the buffer requirement will decrease.

In SFB, the shortest time to download all the blocks occurs when the video segments on all channels are downloaded concurrently. This is equal to the time needed to download the video segment on the last channel as this video segment is the largest. Thus, if we can 'make'

a larger video segment out of the existing video segments without violating the principle of viewing continuity, the shortest-length will increase.

In SFB with K logical channels (where $K > 2$), the video file is equally divided into 2^{K-1} video segment blocks. The number of blocks that made up the video segment on channel i, denoted as $C(i)$, follows the function below:

$$C(i) = \begin{cases} 1 & i = 0, \\ 1 & i = 1, \\ 2 * C(i-1) & 2 \le i \le (K-1) \end{cases}$$

Each video segment is then periodically broadcast on a channel. Figure 6.1 shows the broadcast schedule of SFB with four channels.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8

Figure 6.1. The broadcast schedule of SFB (4 logical channels).

Suppose we now add one more logical channel to the K-channel SFB scheme. This extra channel (i.e., channel K) is also used to broadcast the last video segment but with $C(K-1)/2$ ($= 2^{K-3}$) timeslots difference lagging behind that on channel $(K-1)$. This means that the broadcast of the last video segment on channel K is shifted to right by 2^{K-3} timeslots relative to that on channel $(K-1)$. The broadcast of segments on the existing channels remains the same. We call this new scheme as **Small-Buffer Skip-Forward Broadcasting** (SB-SFB). The broadcast schedule of SB-SFB is shown in Figure 6.2. In the figure, channel 3 and channel 4 broadcast the same video segment but with a timeslot difference of two.

timeslot #	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10	t11	t12	t13	t14	t15
channel 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
channel 1	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
channel 2	3	4	3	4	3	4	3	4	3	4	3	4	3	4	3	4
channel 3	5	6	7	8	5	6	7	8	5	6	7	8	5	6	7	8
channel 4	7	8	5	6	7	8	5	6	7	8	5	6	7	8	5	6

Figure 6.2. The broadcast schedule of SB-SFB (5 logical channels).

With the extra logical channel, the recurring period of the last segment in SB-SFB becomes the same as that of second-last video segment. Let us denote the video segment on logical channel i as S_i. In the broadcast schedule of SB-SFB with $(K + 1)$ channels, there is always a new occurrence of S_{K-1}, either on channel $(K - 1)$ or on channel K, comes after each instance of S_{K-2} on channel $(K - 2)$. As a result, the downloading of S_{K-1} can always start after that of S_{K-2} and the downloading period of them will not overlap. These two segments, S_{K-1} and S_{K-2}, can thus be viewed as a single large segment spreads over two channels.

In SB-SFB with $(K + 1)$ channels, where $K > 2$, clients only need to download video segments from the first $(K - 1)$ channels concurrently in the worst case since the last two channels are both used to deliver S_{K-1}. The shortest time to download all segments is equal to the sum of the delivery time of S_{K-2} and S_{K-1}, i.e., the delivery time of $(2^{K-3} + 2^{K-2})$ blocks. Hence, the client-side buffer requirement (as a fraction of video file size) in SB-SFB is

$$
\begin{aligned}
\textit{Buffer size} &= 1 - \frac{2^{K-3} + 2^{K-2}}{2^{K-1}} \\
&= \frac{2^{K-1} - (2^{K-3} + 2^{K-2})}{2^{K-1}} \\
&= \frac{2^{K-3} * (2^2 - 1 - 2)}{2^{K-1}} \\
&= \frac{2^{K-3}}{2^{K-1}} \\
&= 1/4
\end{aligned}
$$

At the cost of one more logical channel, the buffer requirement in SB-SFB is reduced to one quarter of a video file size, which is a saving of 50% relative to SFB. Besides, the client-side I/O bandwidth requirement in SB-SFB is less than SFB by a bandwidth equivalent to one logical channel.

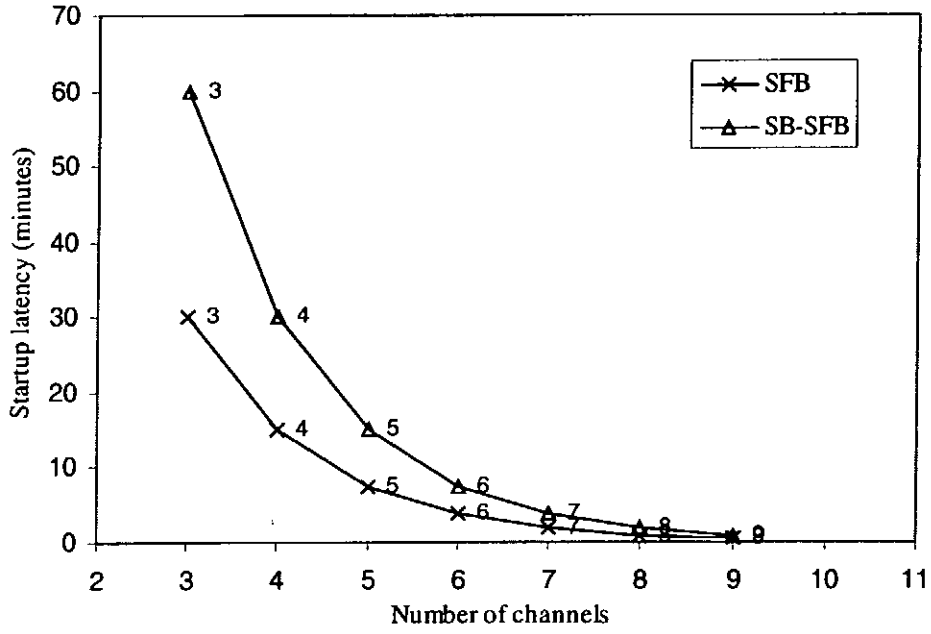

Figure 6.3. The startup latencies in SB-SFB and SFB.

Figure 6.3 shows the startup latencies in SB-SFB and SFB versus the number of logical channels employed for a 120-minute video. The startup latency in SB-SFB is longer than that of SFB under the same number of logical channels, but is equal to that of SFB with one channel less. This is because SB-SFB, segmented a video in the same way as SFB, used one more channel than SFB in order to reduce the client-side buffer requirement. Under the same number of logical channels, the difference in startup latency between the two schemes becomes smaller when the number of channels increases.

6.3 Interactive Operations in SB-SFB

In Chapter 3, we presented how SFB supports skip-forward interaction, on-the-fly seamless channel transition and pause/resume interaction. Since the buffer requirement and the broadcast schedule of SB-SFB are different from SB-SFB, it is necessary to re-evaluate the support of these three features in SB-SFB.

6.3.1 Skip-Forward Interaction

In SFB, the first two video segment blocks (or equivalently the first two video segments) are downloaded concurrently. After that, the consequent blocks begin to be prefetched. In such a download approach, the succeeding block with reference to the current point-of-play is

always prefetched in the buffer or being downloaded (this case only happens in the second block). This is the reason why SFB can provide skip-forward interaction.

In SB-SFB, only the downloading of blocks of the last video segment is changed such that it starts right after the downloading of the last block of the second-last video segment. Since the blocks of the second-last segment are readahead before its playtime, the blocks of the last video segment will also be readahead. As a result, skip-forward interaction is still supported by SB-SFB.

6.3.2 Channel Transition

In SB-SFB with $(K + 1)$ logical channels, the broadcast schedule in the first K logical channels is the same as in SFB. This means that instantaneous seamless channel transitions among broadcast schedules in these K channels are feasible. Therefore, we only need to study the broadcast of the segment on the last channel (channel K).

Denoting S^m_i as the video segment on channel i of SB-SFB with m channels. S^m_{m-2} is the same as S^m_{m-1}. Let us compare SB-SFB with $(K + 1)$ channels and SB-SFB with K channels. Each occurrence of S^{K+1}_K is shifted to the right by 2^{K-3} timeslots with respect to S^{K+1}_{K-1}. On the other hand, each occurrence of S^K_{K-1} is shifted to the right by 2^{K-4} timeslots with respect to S^K_{K-2}. We note that the timeslot duration in the $(K + 1)$-channel SB-SFB scheme is half of that in the K-channel scheme. Thus, in terms of the timeslot duration in the $(K + 1)$-channel scheme, S^K_{K-1} is shifted to the right by

$$2 * 2^{K-4} = 2^{K-3} \text{ timeslots,}$$

which is the same as that of S^{K+1}_K. This means that both schemes broadcast a new instance of segment on the last channel at the same timeslot. In the derivation in Section 3.4.2, it has been shown that S^{K+1}_K is the same as S^K_{K-1}. As a result, when there is a channel transition (step-down or step-up) in SB-SFB between $(K + 1)$ channels and K channels, S^{K+1}_K can be found in the corresponding timeslots in the broadcast schedule of the K-channel scheme or vice versa. Therefore, on-the-fly seamless channel transition can still be launched in SB-SFB.

6.3.3 Pause/Resume Interaction

The capability to support a pause/resume interaction depends on whether the playback duration of the video data in the buffer is long enough to cover the longest recurring period among blocks in the broadcasting scheme. In SB-SFB with $(K + 1)$ logical channels, the total number of blocks is 2^{K-1} and the client-side buffer requirement is one quarter of a video file size, i.e., 2^{K-3} blocks. On the other hand, the longest recurring period among blocks is 2^{K-3} timeslots, which is one quarter of the video length. Therefore, pause/resume interaction is still supported by SB-SFB.

Summing up, the performance of SB-SFB remains the same as SFB even though the buffer requirement is much reduced.

6.4 Summary

While the Skip-Forward Broadcasting scheme can support random forward repositioning, the client-side buffer requirement is still considerable (about half of a video file size). In this chapter, an enhancement to SFB is proposed. Such enhancement can reduce the buffer requirement by half at the expense of one more logical channel allocated to a video, while preserving all the desirable features of SFB. The same enhancement can equally be applied to the Short-Range Fast-Forward Broadcasting scheme. Although this method is a tradeoff between the network bandwidth and the client-side hardware cost, it can be critical to commercial viability when the set-top box's price tag is a key issue to market penetration.

Chapter 7: Conclusions

With the advancement of broadband networking technology, processor speed and disk capacity, Video-on-Demand (VoD) services have become possible. An interactive VoD system can even provide clients with the VCR interactions such as pause/resume and fast-forward. One of the VoD service policies is called broadcasting which aims at efficiently delivering a set of selected popular videos. Earliest periodic broadcast schemes simply broadcast instances of a video overlapped in time. However, the startup latency can be improved only linearly with the increase in the number of instances, or allocated bandwidth. This latency can be reduced exponentially by prefetching video data into the buffer at the client-end. However, none of the schemes have adequately provided solutions to interactive VCR functions. In this report, three new broadcasting schemes, which address the issue of VCR functions in broadcast VoD service, are presented and analyzed.

The first proposed scheme, the Skip-Forward Broadcasting (SFB) scheme, allows a client to reposition his/her point-of-play within a range of a video segment block in the forward direction. In SFB, a video is segmented into uniform video segment blocks that are broadcast on the logical channels operating at the normal video playout rate. The first two video segment blocks are downloaded simultaneously such that there is always at least one prefetched block in the client-side buffer to support skip-forward. Furthermore, SFB supports pause/resume interaction and can provide instantaneous step-up and step-down seamless channel transition on-the-fly.

The second proposed broadcasting scheme, called Short-Range Fast-Forward Broadcasting (SRFFB), supports the fast-forward interaction within a range of two video segment blocks. The scheme design is similar to SFB, except that the first two blocks are grouped into one block which is then transmitted to clients at a rate twice of the video playout rate. By doing so, the clients can fast-forward the video right at the beginning. The subsequent blocks are broadcast on channels at the normal playout rate and are always prefetched into the client's buffer. Similar to SFB, SRFFB also supports on-the-fly seamless channel transition and pause/resume interaction.

The third proposed broadcasting scheme is known as Mirrored-Pyramid Broadcasting (MPB). The main objective of this scheme is to reduce the client-side buffer requirement. Unlike SFB and SRFFB, all channels in MPB deliver video segment at a rate twice of the normal video playout rate. Thus, each segment only needs half of its playback duration to transmit. Moreover, the size of the segments in MPB is first increased geometrically by factors of two, and then shrunk in the reverse rate. As a result, the largest segment is not delivered on the last logical channel, but on the middle. The playback duration of a segment will always cover the time frame occupied by an instance of suitable succeeding segment on the next channel. Therefore, clients need to download video data from at most two channels simultaneously at any one time. This is an advantage over FDB, SFB and SRFFB in which video data needs to be concurrently downloaded from more than two channels. In MPB, the client-side buffer requirement can be lowered to one-third of a video file size.

Although MPB can reduce the client-side buffer requirement, its network bandwidth requirement is larger. Therefore, an Enhanced Mirrored-Pyramid Broadcasting (EnMPB) scheme is proposed to cut the network bandwidth requirement down to about 50% of MPB, without increasing the buffer requirement. In EnMPB, the video segments on the trailing channels are packed into one single large segment which can be broadcast on one channel. As a result, less channels are used in EnMPB and the network bandwidth is saved. EnMPB also supports fast-forward interaction. Moreover, the second and subsequent FF attempts can be performed easier in EnMPB than in SRFFB.

Lastly, a Small-Buffer Skip-Forward Broadcasting (SB-SFB) scheme, which is an improved version of SFB, is proposed. The scheme uses an extra channel to broadcast the timeslot-shifted last video segment of SFB. By doing so, SB-SFB can successfully reduce the client-side buffer requirement in SFB by half while preserving the features of SFB. The same technique can also be applied to SRFFB.

Table 7.1 gives a summary on the performance of all the proposed broadcasting schemes. A selection guide for these schemes is given in Table 7.2.

	Number of Channels	Network Bandwidth **	Client-side Buffer Requirement (% of video file size)	Client-side I/O Bandwidth Requirement **	Startup Latency for an M-minute video (minutes)	Support of Seamless Channel Transition
SFB	$i+1$	$(i+1)*b$	$\leq 50\%$	$(i+1)*b$	$M/2^i$	yes
SRFFB	i	$(i+1)*b$	$\leq 50\%$	$(i+1)*b$	$M/2^i$	yes
SB-SFB	$i+2$	$(i+2)*b$	$\leq 25\%$	$i*b$	$M/2^i$	yes
MPB	$2j+1$	$(2j+1)*2b$	$\geq 33\%$ and $\leq 50\%$	$2*2b$	$\frac{M}{2*(3*2^j-2)}$	no
EnMPB	$j+2$	$(j+2)*2b$	$\geq 33\%$ and $\leq 50\%$	$2*2b$	$\frac{M}{2*(3*2^j-2)}$	no

** b is the video playout rate

Table 7.1. Performance summary.

	Skip Forward	Fast Forward	Pause/ Resume	Channel Transition	Buffer Size	Network Bandwidth	Client's I/O Bandwidth
SFB	✓✓	✓	✓	✓	moderate	small	moderate
SRFFB	✓	✓✓	✓	✓	moderate	small	moderate
SB-SFB	✓✓	✓	✓	✓	smallest	small	moderate
SB-SRFFB	✓	✓✓	✓	✓	smallest	small	moderate
MPB	✓	✓✓			small	large	small
EnMPB	✓	✓✓			small	moderate	small
FDB					moderate	small	moderate
PB					large	moderate	moderate

Table 7.2. Selection guide.

The VCR interactions supported by the three new proposed broadcasting schemes are all about forward interactions. Backward interactions, such as fast-rewind and skip-backward, are not addressed here. To some extent, short-range backward interactions can be supported by extra/optional buffers in the client's set-top box. To support large-range backward interactions, a good client-side buffer manager is needed to decide which old video segment blocks should be removed to provide space for new readahead data, without degrading the backward interaction capability. Because of this, backward interactions call for techniques quite different from that of forward interactions. Further work, therefore, can focus on the implementation of backward interactions in broadcasting schemes. Hopefully, the schemes proposed and analyzed here can provide critical insights to spark ideas on further explorations.

References

[1] Abram-Profeta, E.L. and Shin, K.G. "Providing Unrestricted VCR Functions in Multicast Video-on-Demand Servers". *Proceedings of the 1998 IEEE International Conference on Multimedia Computing and Systems*, Austin, T.X., 28 June-1 July, 1998, pp.66-75 (1998)

[2] Aggarwal, C.C., Wolf, J.L. and Yu, P.S. "Design and Analysis of Permutation-Based Pyramid Broadcasting". *Multimedia Systems*, Vol. 7, No. 6, pp.439-448 (1999)

[3] Almeroth, K.C. and Ammar, M.H. "The Use of Multicast Delivery to Provide a Scalable and Interactive Video-on-Demand Service". *IEEE Journal on Selected Areas in Communications*, Vol. 14, No. 6, pp.1110-1122 (1996)

[4] Dan, A., Dias, D.M., Mukherjee, R., Sitaram, D. and Tewari, R. "Buffering and Caching in Large-Scale Video Servers". Proceedings of the IEEE COMPCON, San Francisco, California, 5-9 March, 1995, pp.217-224 (1995)

[5] Dan, A., Shahabuddin, P., Sitaram, D. and Towsley, D. "Channel Allocation under Batching and VCR Control in Video-on-Demand Systems". *Journal of Parallel and Distributed Computing*, Vol. 30, No. 2, pp.168-179 (1995)

[6] Dan, A., Sitaram, D. and Shahabuddin, P. "Dynamic Batching Policies for an On-Demand Video Server". *Multimedia Systems*, Vol. 4, No. 3, pp.12-121 (1996)

[7] Doganata, Y.N. and Tantawi, A.N. "Storage Hierarchy in Multimedia Servers". In Chung, S.M., ed., *Multimedia Information Storage and Management*, Kluwer Academic Publishers, Boston, pp.61-94 (1996)

[8] Eager, D.L. and Vernon, M.K. "Dynamic Skyscraper Broadcasts for Video-on-Demand", *Proceedings of the Fourth International Workshop on Multimedia Information Systems*, Istanbul, Turkey, 24-26 September, 1998, pp.18-32 (1998)

[9] Eager, D.L., Vernon, M. and Zahorjan, J. "Minimizing Bandwidth Requirements for On-Demand Data Delivery", IEEE Transactions on Knowledge and Data Engineering, Vol. 13, No.5, pp. 742-757 (2001)

[10] Gemmell, D.J. "Disk Scheduling for Continuous Media". In Chung, S.M., ed., *Multimedia Information Storage and Management*, Kluwer Academic Publishers, Boston, pp.1-21 (1996)

[11] Ghose, D. and Kim, H.J. "Scheduling Video Streams in Video-on-Demand Systems: A Survey". *Multimedia Tools and Applications*, Vol. 11, pp.167-195 (2000)

[12] Golubchik, L., Lui, J.C.S. and Muntz, R.R. "Adaptive Piggybacking: A Novel Technique for Data Sharing in Video-on-Demand Storage Servers". *Multimedia Systems*, Vol. 4, No. 3, pp.140-155 (1996)

[13] Hua, K.A., Cai, Y. and Sheu, S. "Exploiting Client Bandwidth for More Efficient Video Broadcast". *Proceedings of the 7th International Conference on Computer Communications and Networks*, Lafayette, L.A., 12-15 October, 1998, pp.848-856 (1998)

[14] Hua, K.A., Cai, Y. and Sheu, S. "Patching: A Multicast Technique for True Video-on-Demand Services". *Proceedings of the Sixth ACM International Conference on Multimedia*, Bristol, United Kingdom, 13-16 September, 1998, pp.191-200 (1998)

[15] Hua, K.A. and Sheu, S. "Skyscraper Broadcasting: A New Broadcasting Scheme for Metroplitan Video-on-Demand Systems". *Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, Cannes, France, 14-18 September, 1997, pp.89-100 (1997)

[16] Juhn L.S. and Tseng, L.M. "Harmonic Broadcasting for Video-on-Demand Service". *IEEE Transactions on Broadcasting*, Vol. 43, No.3, pp.268-271 (1997)

[17] Juhn, L.S. and Tseng, L.M. "Staircase Data Broadcasting and Receiving Scheme for Hot Video Service". *IEEE Transactions on Consumer Electronics*, Vol. 43, No.4, pp.1110-1117 (1997)

[18] Juhn, L.S. and Tseng, L.M. "Fast Data Broadcasting and Receiving Scheme for Popular Video Service". *IEEE Transactions on Broadcasting*, Vol 44, No.1, pp.100-105 (1998)

[19] Jun, S.B. and Lee, W.S. "Video Allocation Methods in a Multi-Level Server for Large-Scale VoD Services". *IEEE Transactions on Consumer Electronics*, Vol. 44, No. 4, pp.1309-1318 (1998)

[20] Liao, W.J. and Li, V.O.K. "The Split and Merge Protocol for Interactive Video-on-Demand". *IEEE Multimedia*, Vol. 4, No. 4, pp51-62 (1997)

[21] Lin, F.Y.S. "Optimal Real-time Admission Control Algorithms for the Video-on-Demand (VoD) Service". *IEEE Transactions on Broadcasting*, Vol. 44, No. 4, pp.402-408 (1998)

[22] Little, T.D.C. and Venkatesh, D. "Prospects for Interactive Video-on-Demand". *IEEE Multimedia*, Vol. 1, No. 3, pp.14-23 (1994)

[23] Mundur, P., Simon, R. and Sood, A. "Integrated Admission Control in Hierarchical Video-on-Demand Systems". *Proceedings of the 1999 IEEE International Conference on Multimedia Computing and Systems 1999*, Florence, Italy, 7-11 June, 1999, pp.220-225 (1999)

[24] Özden, B., Rastogi, R. and Silberschatz, A. "On the Design of a Low-Cost Video-on-Demand Storage System". *Multimedia Systems*, Vol. 4, No. 2, pp.40-54 (1996)

[25] Pâris, J.F., Long, D.D.E. and Mantey, P.E. "Zero-Delay Broadcasting Protocols for Video-on-Demand". *Proceedings of the 1999 International Conference on Multimedia*, Orlando, FL USA, 30 October – 5 November, 1999, pp.189-197 (1999)

[26] Poon, W.F., Lo, K.T. and Feng, J. "Multicast Video-on-Demand System with VCR Functionality". *Proceedings of the 1998 International Conference on Communication Technology*, Beijing, China, 22-24 October, 1998, pp.S23-10 (1998)

[27] Poon, W.F. and Lo, K.T. "New Batching Policy for Providing True Video-on-Demand (T-VoD) in Multicast System". *Proceedings of 1999 IEEE Interactional Conference on Communications*, Vancouver, Canada, 6-10 June, 1999, pp.983-987 (1999)

[28] Rangan, P.V and Vin, H.M. "Efficient Storage Techniques for Digital Continuous Media". *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, pp.564-573 (1993)

[29] Ruemmler, C. and Wilkes, J. "An Introduction to Disk Drive Modeling". *IEEE Computer*, Vol. 27, No. 3, pp.17-28 (1994)

[30] To, J.T.P. and Hamidzadeh, B. "Dynamic Real-Time Scheduling Strategies for Interactive Continuous Media Servers". *Multimedia Systems*, Vol. 7, No. 3, pp.91-106 (1999)

[31] Tseng, Y.C., Hsieh, C.M., Yang, M.H., Liao, W.H. and Sheu, J.P. "Data Broadcasting and Seamless Channel Transition for Highly-Demanded Videos". *Proceedings of IEEE INFOCOM 2000*, USA, 26-30 March, 2000, pp.727-736 (2000)

[32] Viswanathan, S. and Imielinski, T. "Metropolitan Area Video-on-Demand Service Using Pyramid Broadcasting". *Multimedia Systems*, Vol. 4, pp.197-208 (1996)

[33] Vin, H.M., Goyal, A., Goyal, A. and Goyal, P. "An Observation-Based Admission Control Algorithm for Multimedia Servers". *Proceedings of the 1994 IEEE International Conference on Multimedia Computing and Systems*, Boston, M.A., 15-19 May, 1994, pp.234-243 (1994)

[34] Wong, W.T., Zhang, L. and Pang, K.K. "Video on Demand Service Policies". *Proceedings of IEEE Singapore International Conference on Networks*, Singapore, 3-7 July, 1995, pp.560-564 (1995)

[35] Wu, C.S., Ma, G.K. and Liu, M.C. "A Scalable Storage Supporting Multistream Real-Time Data Retrieval". *Multimedia Systems*, Vol. 7, No. 6, pp.458-466 (1999)

[36] Yu, P.S., Wolf, J.L. and Shachnai, H. "Scheduling Issues in Video-on-Demand Systems". In Chung, S.M., ed., *Multimedia Information Storage and Management*, Kluwer Academic Publishers, Boston, pp.183-207 (1996)

Publication List

1. Ma, H.S. and To, T.P.J. "Mirrored-Pyramid Broadcasting for Metropolitan On-Demand Video Delivery". *Proceedings of the 2000 Workshop on Multimedia Data Storage, Retrieval, Integration and Applications*, Hong Kong, 13-15 January, 2000, pp.104-109 (2000)

2. Ma, H.S., To, T.P.J. and Lun, P.K. "Enhanced Mirrored-Pyramid Broadcasting for Video-On-Demand Delivery". *Proceedings of the 2000 IEEE Asian Pacific Conference on CAS*, Tianjin, China, 4-6 December, 2000, pp.875-878 (2000)

3. Ma, H.S., To, T.P.J. and Li, C.K. "A New Broadcasting Scheme Supporting Fast-Forward for Video-on-Demand Service". *Proceedings of the International Network Conference 2002*, Plymouth, UK, 16-18 July, 2002, accepted (2002)